Project no. 027087

TENCompetence

Building the European Network for Lifelong Competence Development

Project acronym: Integrated Project TENCompetence

Thematic Priority: 2.4.10

D6.1 – ANNEX 3 - Runtime Systems

Due date of deliverable: 31-05-2007
Actual submission date: 03-01-2008

Start date of project: 01-12-2005 Duration: 4 years

University of Bolton

Version 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Building the European Network

For Lifelong Competence Development

TENCompetence IST-2005-027087

D6.1 Annex 3 - Runtime Systems

Work package	Work package 6, Learning activities and units of learning
Task	Task 6.1, Task 6.2, Task 6.3 and Task 6.4
Date of delivery	**Contractual**: 31-05-2007 **Actual**: 03-01-2008
Code name	**Version**: 1.0 Draft ☐ Final ☒
Type of deliverable annex	Report
Security (distribution level)	Public
Contributors	David Griffiths, Tom Franklin, Paul Sharples, Phillip Beauvoir, Scott Wilson, Yongwu Miao, Colin Tattersall, Rob Koper, Krassen Stefanov, Milen Petrov, Adelina Aleksieva - Petrova, Toni Navarrete, Helena Batlle, Demetrios Sampson, Pythagoras Karampiperis, Judith Schoonenboom, Marco Luccini, Linda Napoletano
Authors (Partner)	UB, OUNL, SU, FBM-UPF, CERTH/ITI, UvA, ILABS
Contact Person	David Griffiths (UB)
WP/ Task responsible	David Griffiths (UB)
EC Project Officer	Mr. Martin Májek
Abstract (for dissemination)	This Annex provides detailed information on work carried out on runtime systems during the first 18 months of the TENCompetence project
Keywords List	IMS Learning Design Authoring, Assessment Model, Run-time IMS LD Services, Pedagogical Templates, Units of Learning

TENCompetence Project Coordination at: Open University of the Netherlands

Valkenburgerweg 177, 6419 AT Heerlen, The Netherlands

Tel: +31 45 5762624 – Fax: +31 45 5762800

TABLE OF CONTENTS

1. Summary of this Annex

The TENCompetence project has a strategy of using Open Source and standards compliant systems. Task 3 of WP6 is responsible for providing effective runtime systems which correspond to these requirements. In practice this means building on the infrastructure currently available for running IMS specifications.

There are three main threads of work described here.

Firstly, IMS LD runtime. Experience in previous projects had identified one of the key shortcomings for IMS LD runtime as being the lack of runtime services. **Section 2 describes the state of the art** at the start of the project, and identifies a strategy for integrating SCORM and IMS LD. These are two of the most significant specifications for "learning objects", which are often seen as being incompatible alternatives. **Section 3 provides details of the integration work carried out**, testing and demonstration. This work was successful and effective, but also required a substantial amount of developer time to carry out, and is rather inflexible. For that reason it does not constitute an ideal solution for future project work on IMS LD runtime. For **downloads and a demonstrator**, please also see http://www.tencompetence.org/ldruntime/.

Secondly, IMS QTI runtime. The assessment strategy of the TENCompetence project requires IMS QTI runtime. As there is no runtime implementation of the latest version of the specification (v2.1) there was a need for the project to fill this gap. **Section 5** describes work in **updating the APIS QTI runtime system** to meet the requirements of QTI 2.1. The **source code for QTI 2.1 runtime** is available at http://sourceforge.net/projects/newapis.

Thirdly, the connection protocol. Work on this task not only has to produce effective systems, it also has to identify generalisable solutions. In parallel with the implementation work, desk research was carried out into ways in which services could be provided in a more flexible way through a connection protocol, and section **6 sets out the approach to IMS LD service integration which was developed**. Section 7 provides a first version of the protocol, including a description of the requirements and services. This is currently being implemented in the second period of work on this task.

2. Introduction to runtime work in WP6

In the work reported in this section, we are addressing a number of difficult issues related to runtime services for the Units of Learning and Units of Assessment developed in the project. As described in the main text of this deliverable, Tasks 1 and 2 deal with the authoring process for learning activities and assessments respectively. The outputs of this process are Units of Learning (which are expressed in IMS LD) and Units of Assessment (which are expressed as a combination of IMS LD and IMS QTI). For this approach to be effective there must be run-time systems which interpret these documents and coordinate the appropriate activities for learners. The Run-time Component task sets out to build on existing Open Source runtime engines to ensure that effective and usable systems are available to meet this need. Two principal challenges were identified:

a) There is no runtime environment which is compliant with the latest version of the IMS QTI specification, version 2.1.

b) There is a lack of flexible runtime services (e.g. a forum, or a chat, run external learning objects and tests, etc.) which can be launched from an LD player (typically a system based on the CopperCore LD Engine).

Both these issues have been addressed in this first phase of project work.

Firstly, work to update runtime services for QTI is described in Section 4 of this Annex. The choice of an approach to resolving this issue was not difficult, as the only existing Open Source runtime environment for IMS QTI is APIS. This was developed with funding from the UK eFramework and has been extended in this project reporting period to support IMS QTI 2.1. The principal challenges to be overcome involved the way in which the new architecture of QTI 2.1 should be handled, and how the new item types should be represented. This was a major undertaking as the structure of the specification has been radically changed in V. 2.1. At a higher level, however, integration with IMS LD runtime is relatively straightforward, because QTI has already been integrated with IMS LD runtime, in the SLeD Learning Design player. The TENCompetence version of APIS is available at http://sourceforge.net/projects/newapis.

However, the fact that QTI has been ingenerated with IMS LD makes it an exception in IMS LD runtime. We now discuss the general issues of integrating runtime services, in order to set the scene for the description of the services integration work carried out.

ELearning interoperability initiatives have developed as more or less coordinated explorations of various way of achieving more effective applications of Information Technology in education. There has not been an overall plan governing the way in which the domain of eLearning has been divided into areas, with each covered by a specification. Rather we have a collection of specifications, each of which responds to a need identified at the time when work on it was commenced. Each specification is explicitly or implicitly informed by theories, ideas and assumptions, which may be different or conflicting for the various specifications. Moreover, specifications are developed within a particular technological context, and as this context changes so do the consequences of using any given specification.

In eLearning today there are a relatively small number of successful interoperability specifications, many of them produced or adopted by IMS. They are the product of a great deal of effort and reflection, and a substantial amount of implementation has been carried out to support their use. This is a very valuable resource for eLearning, but

the lack of an integrating framework means that it is hard for them to work together. One of the reasons why SCORM is important is that it performs this function of bringing together a number of IMS Specifications into an application profile, and promotes the implementation of software to run them.

the varied perspectives which inform the specifications have implications for their use. Thus SCORM provides a valuable integrated infrastructure, but it is not capable of running sophisticated learning scenarios represented in IMS LD. Nor can IMS LD systems interoperate with SCORM objects.

Regarding the second point, IMS LD was developed by taking the OUNL Educational Modelling Language and adapting it (EML). In this process a number of the services provided by EML were removed, for two reasons:

In order to avoid duplication those parts of EML which addressed areas already covered by other IMS specifications were removed from the IMS LD specification. This was particularly the case for the generation of tests and questions (covered by IMS QTI)

In order to make the IMS LD specification more attractive to implementers, and to ensure that the threshold for interoperability was not too high, only a small number of simple services were included (send mail, monitor, index search). As a result a number of the integrated services in EML were removed in IMS LD (for example forums).

In TENCompetence we now have to deal with the consequences of these decisions made by IMS in the development of their specifications. The decisions were no doubt reasonable at the time, as it was a priority to develop usable specifications which would be attractive to adopters. Nevertheless the issues of interoperability between specifications and the paucity of services in IMS LD are now pressing concerns which are limiting the usefulness of the IMS suite of specifications, and hence the whole vision of eLearning based on interoperability specifications. In TENCompetence we are seeking solutions to this problem, in the first instance for the system under development, but also as a general solution for the wider context of eLearning specifications. Consequently the work we report here is not simply a technical solution to a particular implementation problem, but rather the initial results of the first serious attempt to resolve the contradictions which are inherent in the currently available set of eLearning specifications.

In this section of the milestone we first we go into greater detail on the issues which surround the use of services in IMS LD. Next we describe some of the ways which these issues are being addressed by IMS and other initiatives, which propose frameworks which could enable new solutions to the problem. This provides a wider context in which our work can be situated, but none of these initiatives is likely to produce workable solutions for TENCompetence in the short or even medium term. Consequently we move on to discuss practical steps which can be taken to implement services within the IMS

LD. Our starting point in this is work done by the Valkenburg group in establishing an architecture for the implementation of IMS LD, and in particular the work of Bill Olivier, who proposed a programme of work which has informed the approach taken by TENCompetence .

A key task in this work package is to establish a new specification for communicating between IMS LD runtime systems and services. The need for this has been established in the literature (see Bill Olivier below), but no practical work had been done to show the real needs of users in practice. Consequently the Project adopted a two pronged approach. On the one hand we worked on building a functioning system which enables IMS LD runtime to fully integrate Sharable Content Objects from the SCORM 1.2 spec. The first phase of this work was completed in the final quarter of 2006, and the demonstrator is now available. The experience of carrying out this development has shown the scale of the problem, even in this quite restricted use case. Thus the implementation work is not only intended to provide valuable functionality which can be used within the TENCompetence framework, but also to provide the basis for the identification of requirements for the connection protocol which will be a key output from our work.

The documentation for the SCORM integration discusses the background behind the task and how existing tools might be reused together. It then explains some of the issues around how to achieve integration between them, and progresses through to design issues and implementation. There is also a brief technical section on how integration was finally achieved. We will then cover the way in which an IMS LD Unit of Learning can be authored so that it can contain ADL SCORM content. Finally we will progress to a section outlining how this would look to a user and explain the runtime processes involved.

2.1. Will the real service please stand up?

In the olden days, it was the word ‘object’ which caused difficulties. Nowadays it’s ‘service’. We have web services, semantic web services, SOAs, ELF services…. and we also have IMS LD’s Services. The latter form the starting point for this work – ‘web services’ are from this viewpoint infrastructural elements which could be used in the implementation of IMS LD’s services. IMSLD offers a set of services which can be included in learning process descriptions. Designers indicate which services are available at which point in the flow. The IMSLD services are:

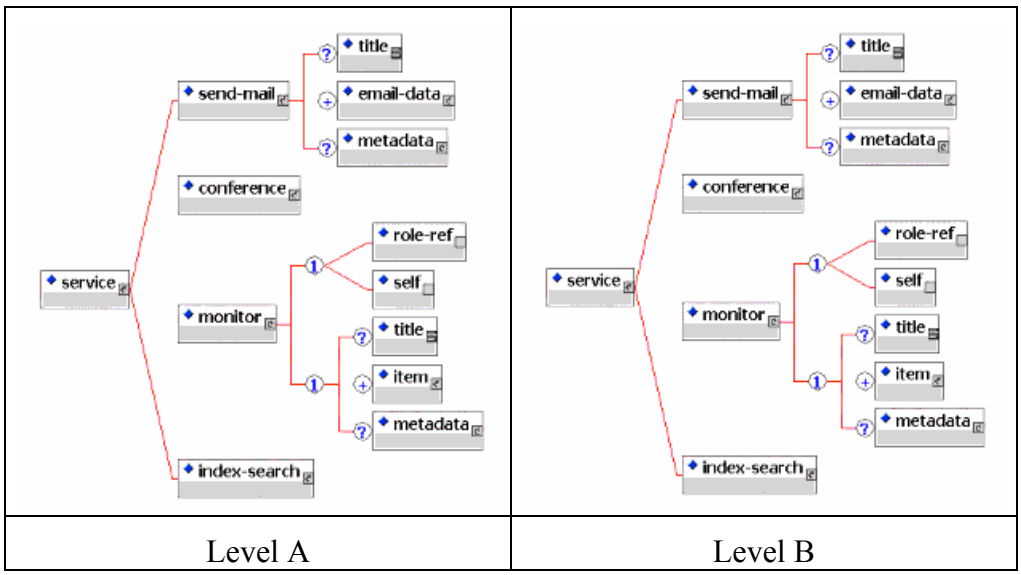


Figure 1: IMS LD Level A and Level B Services

Selected remarks on IMSLD services below (IMS a 2003):

- *Mail*: Fixed choice: 'all-persons-in-role' or 'persons-in-role'. With the first choice, the user agent only allows messages to be sent to the role, indicating that all persons in the role get the message. With the second choice, the user agent allows a user to select one or more individuals within the specified role to send the message to.
- *Conference*: The elements participant, observer, conference-manager, moderator facilitate the setting of the user rights in the conferences. They each contain a role-ref which associates them with a role in the learning design. If more than one role is to be assigned to a conference role (e.g. several LD roles are to be participants) then several instances of the conference role are needed, one for each LD role. It depends on the implementation how a conference is set up and managed: 1. When the conference system is an integral part of the runtime system, it is expected to be set up automatically; 2. When the conference is external, the user-rights can be set manually by the conference manager. The conference manager must be able to get a list from the runtime agent about which conferences of what type, for what users, with what rights, need to be set up. 3. Using the data in this conference element, the conferences can also be set up by a generating scripts, configuration files or a legacy interface to the rights management system of the conferencing system. In all instances the runtime system must be able to provide this information in a structured way. The item element refers to the resource where the conferencing system is to be found or identified. External conferencing systems can be of any kind accessible through the internet (resource type is webcontent). Examples: netmeeting, placeware (synchronous), first-class, lotus notes, news groups (asynchronous). An announcement object sets the rights: creator of announcement = participant. Reader of announcements = observer.

- *Index-search*: A choice of elements to specify indexing aspects, used to set up a search service. The index is made in the background (not visible to users). The visibility is determined with the search element. The functionality of the index is dependent on the search element: - When search is free-text-search, then the index is made on the resource pointed at in the index (i.e. the underlying html texts). - When search is index-with/without-reference, than only an index is made of the elements which share the same class, including underlying items. This has the form of a table of content.
- *Monitor*: The monitor service provides a facility for users to look at their own properties or that of others in a structured way. A monitor service uses global properties in resources of type 'imsldcontent' to view the properties of one-self or of all users in a role.

The IMS Learning Design specification also notes:

- A service relates to a concrete service facility available at runtime. During design a service has no URL assigned to it, but must be given a URL when the Learning Design is instantiated at runtime. Examples of a Service include a discussion forum, chat rooms, monitoring tools, search facilities, etcetera. In Learning Design the conditions for setting up a service at runtime are specified at an abstract level. For example, for discussion groups it specifies which learning design roles have what type of access (participant, observer, moderator, etc.). Note: if a discussion forum is to be used within a learning design, were it given a predefined URL, then all instances of the Unit of Learning that includes the learning design, wherever and whenever instantiated, would have the same one specific discussion forum.
- A runtime facility (or a human) can setup the necessary facility according to the requirements. In the learning design specification, the abstract declaration of a service facility is called a 'service'. The instantiation of a service is called a 'service facility'.

Additionally, apart from the services explicitly mentioned in the specification, there is another type of service to be handled during execution of a Unit of Learning. By default, an IMSLD can specify content as either ‘webcontent’ or imsldcontent’. However, there may also be the need to integrate other types of content, based on other specifications such as IMSQTI or ADL SCORM. In such cases, the Learning Design runtime environment may not be able process the content and may need to call a third party service with can handle the content. Later in this document we outline the processes involved in developing such a service which allow the Learning Design runtime system process AD SCORM content.

How do these concepts fit into IMSLD?

- A design might have learners operating independently, but require some triggers for them to carry out activities. The mail service could be used to alert learners by having an email be delivered to e.g. Outlook (note the relationship here to level C's notification concept)
- Learners might be split into two groups, one 'for' compulsory vaccination of birds and fowl, one 'against'. The groups would be instructed to use their chat tools (e.g. MSN Messenger) to come up with arguments and counter arguments which are posted in a forum shared by both groups.
- The *What is Greatness* example (Tattersall C, 2004) gives an example of the user of the monitor by both staff and Learners.

2.2. Related work

As background work on the design of a new communication protocol, it makes sense to first to investigate other work in this area. The next section lists work that is related in this area and gives a brief description of what they cover.

1. *The IMS Tools Interoperability Specification (IMS c 2006)*

- The IMS Tools Interoperability (TI) approach addresses the growing demand for a reusable mechanism for integrating third-party tools with core LMS platforms. Tools can add specialist functionality to the LMS such as assessment or discipline-specific teaching aids. The approach recommended greatly simplifies this task whilst also offering a Web Services solution equally applicable to Java and .Net implementers. The reuse of a commonly understood approach across tools will eliminate the need for bilateral solutions, thus focusing investment on adding real value to the learner experience.
- Whilst working on this approach in the IMS Global Learning Consortium, the participants also implemented a demonstrator for alt-i-lab 2005. Their implementations made use of the WSDL auto-generation tool developed by the IMS General Web Services group. The use of this tool has allowed the TI approach to be specified in UML, from which the tool produces a WSDL file which can be used with a variety of Web Services development environments. The fact that there exist working systems that have been publicly demonstrated prior to the release of this document will hopefully install confidence in the approach for other adopters and implementers.
- User authentication is handled by the LMS in each case, whilst the LMS authenticates itself to the tool using a shared secret. The approach exploits a modular context profiling mechanism to pass additional information to the tool.
- The LMS can include the user's ACCLIP profile allowing the tool to self-configure its user interface to the learner's precise needs.

- The optional Outcome profile states the results format required - currently a simple score, but could be HRXML, QTI Results, etc.
- The approach has been designed to allow additional context profiles to be added in the future without impacting currently supported interoperability

2. *IMS Shareable State Persistence (IMS b 2006)*

- The Shareable State Persistence specification describes an extension to e-learning runtime systems (e.g., SCORM) that enables the storage of and shared access to state information between content objects. There is currently no prescribed method for a content object to store (arbitrarily complex) state information in the runtime system that can later be retrieved by itself or by another content object. This capability is crucial to the persistence of the sometimes complex state information that is generated by a variety of interactive content (e.g., simulations) and that is currently stored and retrieved in proprietary formats and through proprietary methods.

3. *Business Process Execution Language for Web Services version 1.1 (IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, 2005)*

- Business Process Execution Language for Web Services provides a means to formally specify business processes and interaction protocols.
- BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions. BPEL4WS defines an interoperable integration model that should facilitate the expansion of automated process integration in both the intra-corporate and the business-to-business spaces.

4. *Learning Design Engines as Remote Control to Learning Support Environments (Harrer. A., Malzahn, N., Hoeksema K , Hoppe U. 2005)*

- We propose an approach that aims at a clear separation of the learning design engine, the specification of the learning flow (as LD documents) and learning environments. According to its current state, the engine controls the learning environment with events (such as "start a new phase"), defined as a vocabulary for a set of environments, that are mapped to the environments' existing functionality (such as "create new workspace"). Thus the engine remotely controls the learning tools while the tools can initiate state transitions in the engine on specific events in the tool.

5. *Collaborative Learning Systems Using IMS-LD and Grid Services (Bote-Lorenzo M.L, Hernández-Leo D, Dimitriadis Y.A. et al, 2004)*

    - CSCL applications are complex distributed systems that pose special requirements towards achieving success in educational settings. Flexible and efficient design of collaborative activities by educators is a key precondition in order to provide CSCL tailorable systems, capable of adapting to the needs of each particular learning environment. Furthermore, some parts of those CSCL systems should be reused as often as possible in order to reduce development costs. In addition, it may be necessary to employ special hardware devices, computational resources that reside in other organizations, or even exceed the possibilities of one specific organization. Therefore, the proposal of this paper is twofold: collecting collaborative learning designs (scripting) provided by educators, based on well-known best practices (collaborative learning flow patterns) in a standard way (IMS-LD) in order to guide the tailoring of CSCL systems by selecting and integrating reusable CSCL software units; and, implementing those units in the form of grid services offered by third party providers. More specifically, this paper outlines a grid-based CSCL system having these features and illustrates its potential scope and applicability by means of a sample collaborative learning scenario.

6. *Web services on demand: WSLA-driven automated management (Dan A, Davis D, Kearney R, Keller A, King R, el al, 2004)*

    - Success of dynamic outsourcing and quickly forming new business relationships are, however, dependent on three critical factors. First, to meet interoperability requirements, access to services needs to be based on open and emerging standards for enabling the service-oriented architecture (SOA) model, and in particular Web and grid services. These services may span a wide range of the outsourcing spectrum, including access to business applications, such as financial services, human resources (HR), and enterprise resource planning (ERP), and infrastructural resources, such as storage, computing resources, and application-hosting platforms. Second, the decision to outsource a part of the business process or application is critically dependent on whether a business partner can be trusted to provide an on-time reliable service. To ensure this quality of service, the service client jointly with the service provider should define a service level agreement (SLA) as a part of a service contract that can be monitored by one or both parties. The same service may be offered at different service levels (in terms of responsiveness, availability, throughput) and priced accordingly. Third, to provide fine-grained outsourcing in a cost-effective and on-time manner, it is essential to support automated management of the entire life-cycle of the business relationship: creation of service offering, creation of SLAs with possible negotiation, provisioning of applications and environments, and monitoring of SLAs both for dynamic allocation of resources and for compliance. To facilitate

this automated management, the SLAs and other agreements need to be specified in machine-executable forms.

7. *Semantic Web Services Framework (SWSF) Overview (Battle S, Bernstein A, Boley H, Grosof B, Gruninger M, et al, 2005)*

   - The promise of Web services and the need for widely accepted standards enabling them are widely recognized, and considerable efforts are underway to define and evolve such standards in the commercial realm. In particular, the Web Services Description Language (WSDL) [WSDL 1.1] is already well established as an essential building block in the evolving stack of Web service technologies, and is being standardized in the W3C's Web Services Description Working Group. WSDL, in essence, allows for the specification of the syntax of the input and output messages of a basic service, as well as other details needed for the invocation of the service. WSDL does not, however, support the specification of workflows composed of basic services. In this area, the Business Process Execution Language for Web Services (BPEL4WS) [BPEL 1.1], under development at OASIS, has the most prominent status. The Choreography Description Language under development by W3C's Web Services Choreography Working Group, serves to "define from a global viewpoint ... the information exchanges that occur and the jointly agreed ordering rules that need to be satisfied" in carrying out a Web service-based transaction [WS-Choreography]. With respect to registering Web services for purposes of advertising and discovery, Universal Description, Discovery and Integration (UDDI) [UDDI v3.02] has received the most attention to date. Standards are also being developed in connection with various other aspects of Web service provisioning, such as reliable messaging, security, and resource management.
   - At the same time, recognition is growing of the need for richer semantic specifications of Web services, based on a compressive representational framework that spans the full range of service-related concepts. Such a framework will enable fuller, more flexible automation of service provision and use, support the construction of more powerful tools and methodologies, and promote the use of semantically well-founded reasoning about services. Because an expressive representation framework permits the specification of many different aspects of services, it can provide a foundation for a broad range of activities, across the Web service lifecycle. For example, richer semantics can support greater automation of service selection and invocation; automated translation of message content between heterogeneous interoperating services; automated or semi-automated approaches to service composition; more comprehensive approaches to service monitoring and recovery from failure; and fuller automation of verification, simulation, configuration, supply chain management, contracting, and negotiation for services.

2.3. Providing support for services within TENCompetence

Essentially, we want TENCompetence to lead to an infrastructure where users *turn-on*, *tune-in*, *learn about*. They open their laptop and the wireless infrastructure automatically hooks them into a network in which they are immediately able to study, discuss, train, play, watch, listen, talk, share, critique.

We have not yet achieved this, but our work in the first year of the project is a significant step in this direction, and contributes to providing an infrastructure which addresses some of today's questions:

1. Where do I plug in my chat tool?
2. Half of my learners use MSN Messenger and the other half AOL Messenger
3. What about Wikis? Shared Spaces, CSCL tools?
4. I have on online assessment service. I want to use that as an LD service
5. I want to integrate an online game into my design.
6. I have a cohort of 2000 Learners and I'd like them to work in groups of 20, each with their own forum.
7. I need to get the transcript of the chat facility back as a post into a forum
8. I need to know whether each learner has posted in the forum

In approaching these problems we base our approach on previous work on LD implementation. The key reference in this respect is Chapter 2 in the Springer Learning Design book by Bill Olivier (Koper, R; Tattersall, C. Eds., 2005). Technical Director of JISC, Bill Olivier's work has informed the e-Framework, which is the most significant initiative for the implementation of a service oriented approach to eLearning, and in which IMS LD has a significant role. The following extract from the chapter represented the starting point of the work reported in this milestone.

> *The selection of services reflects the **most widely implemented** and used services in online learning environments at the time of approval of version 1 of the LD specification: send-mail, conference, monitor, and index search. These **services must either be provided by the player, or be separate services that are linked to by the player** (e.g. they might be provided by standard email and Netnews servers respectively)…. When a UOL is being set up prior to a run with a particular group of participants, the participants have first to be mapped to the roles specified in the learning design. Typically this would be done through a **management utility** provided with the runtime system.*
>
> *The **learning design is then scanned** for all learning services and, with a list of participants for each role, a dedicated instance of the service is set up using the list of participants in the relevant roles and the mapping of LD roles to the service roles contained in the UOL's service definition. Setting up the service can be done in one of two ways. If the service only has a user interface for creating instances, then setting up the service with the actual participants has to be done manually. In this case, the set-up*

function of the management utility should produce a human-readable list of the necessary services together with a list of people mapped to the service's roles. If, on the other hand, the service has a machine-to-machine interface, then the management utility can ***produce a script to automate the process*** *of setting up the service. The ability to set up collaborative and other services automatically is of some practical importance, as without it, the load on system administrators will result in limiting the use of such services and hence conflict with the learning goals.*

Once a service has been set up, the link (URL or other identifier) to this ***service has to be passed back to the player****, along with the reference to the service in the learning design. From then on, the LD player can treat a learning service in the same way as a learning object, by simply providing, at the appropriate point, a hyperlink to it in the learner's web browser interface.*

It is worth noting that where a service such as a conference is requested, it could be met in several ways. One of the systems available where the design is deployed could be used, or this approach could be substituted for a face-to-face meeting or a conference call with a link being made to a web page providing information about time, place, phone number and other details as appropriate.

It should also be noted that services such as computer-based conferencing systems do not have a standardised configuration interface. This means that ***LD management utilities are likely to produce some XML files****, which will then need a* ***further specialised transformation*** *into the configuration calls needed for the particular service to be used. It will be of benefit if all LD management utilities produced such service configuration information in the same XML format, so* ***a small 'adjunct' specification*** *outlining this may well be produced. This would at least limit one side of the many-to-many translations that are otherwise necessary so that only one transformation needs to be written for any given service which all LD management utilities can use. In the longer term, a standard interface to the service may be produced for each service so that the ideal of* ***plug-and-play between LD systems and services*** *can be achieved.*

Learning Services are a significant area that LD opens up, but that is as yet relatively undeveloped, both in the specification and in current LD practice.

Clearly many ***more services could be added to the LD specification****, and it is desirable that they should be, from chat, instant messaging and white-boards, through virtual classrooms and more sophisticated collaborative services, such as virtual design environments, to sophisticated simulation and multi-user game-playing systems.*

The key issue that needs to be addressed is how to add services in such a way that learning designs that use them still ***retain a reasonable degree of portability*** *across different LD-compliant platforms. If all the above services were included, could any system be expected to be compliant? Or should the specification stick to the lowest common denominator for services, as in LD v.1.0, only supporting them as they become commonly available in systems?*

Clearly individual institutions could extend the specification to support their own services, though they would have to adapt their LD instantiation facilities in order to integrate them.

In the meantime, this is an area that is likely to see different communities create applications profiles and optional extensions (i.e. optional for LD system implementers). The application profiles should enable both content and systems to be clearly described

so that the requirements of the one and the capabilities of the other can be determined at a glance.

*One hopeful avenue will be that many of these services will come to be provided by standalone **services**, rather than integrated into increasingly strained Learning Management Systems (LMSs) and Virtual Learning Environments (VLEs). Such loose integration would be facilitated by both **configuration and service interfaces** along the lines being developed by OKI (2004) and IMS. This would allow the addition of services to become independent of particular LMS/VLE providers, but presupposes the availability of at least one instance of any such service, whether open source or commercial, for each service defined, so that anyone could make use of a service specified in a learning design.*

Learning services are likely to come in two varieties: those that are available as downloadable software, either open source or commercial, which are set up as part of a local environment; and those that are set up as remote web services, which again would be either freely accessible or available on a commercial basis.

*To further this approach, it would be desirable to have a registry of learning services, giving their type and the service interface they used, perhaps together with an Open Service Interface Definition (**OSID**) type of adaptor that could be downloaded.*

The way forward

As part of the reviewing process of current specifications and related work, it was decided that it would make sense to investigate how currently one would integrate a new service into a Learning Design runtime system. CopperCore (Vogten H., Martens H., Koper R., 2006) Learning Design Engine has been around for more than two years. Since its inception, the software has grown to handle more and more complexity. It currently is at version 3.0 and boasts the ability to have new services connect to it via one of its newer components, CopperCore Service Integration (C.C.S.I.) The next section explores the integration of one such service and how the process of writing a new service is currently accomplished with the existing framework.

3. Integration work carried out

3.1. Introduction

For **software downloads and a demonstrator** relating to the work reported in this chapter, please see http://www.tencompetence.org/ldruntime/.

Work on CopperCore and SCORM directly responds to the Description of Work document concerning WP6 tasks, namely task two…

*Adapt existing open source authoring and publishing tools, specifically Reload, ASKLDT**, CopperCore, Reload SCORM player**, and CopperAuthor, UPF-QTI-tools to function as a component and service into the TENCompetence System (activity 6.1.1.2). It should allow the effective and efficient implementation of the required pedagogical models.*

(DOW page 62)

*During the first 18 months activities will concentrate on **CopperCore, Reload (LD & SCORM)**, ASK_LDT, CopperAuthor and UPF QTI tools.*

(DOW page 63)

Task two work concerns the selection and integration of software toolsets for inclusion into the TENCompetence Framework. This can be divided into authoring tools and runtime tools. This section concerns runtime and as such attempts to show which specific tools we adopted to use and adapt for this work during this initial period.

3.2. Background

WP6 is geared towards the general theme of Learning Activities & Units of Learning. As a starting point for the runtime work and bearing the above in mind, two existing tools were identified as being suitable for integration.

Coppercore

CopperCore, (Vogten H., Martens H., Koper R., 2006) seemed to provide the starting point for this work. The CopperCore software can be subdivided into three sections.

(a) The Learning Design Engine software allows IMS Learning Designs to be imported, read and then executed. An archive file (zip) containing an IMS Learning Design package can be inspected, disaggregated and parsed for meaning. The constituent sections are then all stored so that a runtime piece of software can access it.

(b) The actual "runtime" section of the process is handled by what is referred to as a "player". This is the software which allows the output from the Learning Design Engine to be formatted and presented to the user. The user can experience how the Unit of Learning "played" using a web browser. The CopperCore software also includes a player

which can be used to view the Unit of Learning. (although there are other players available).

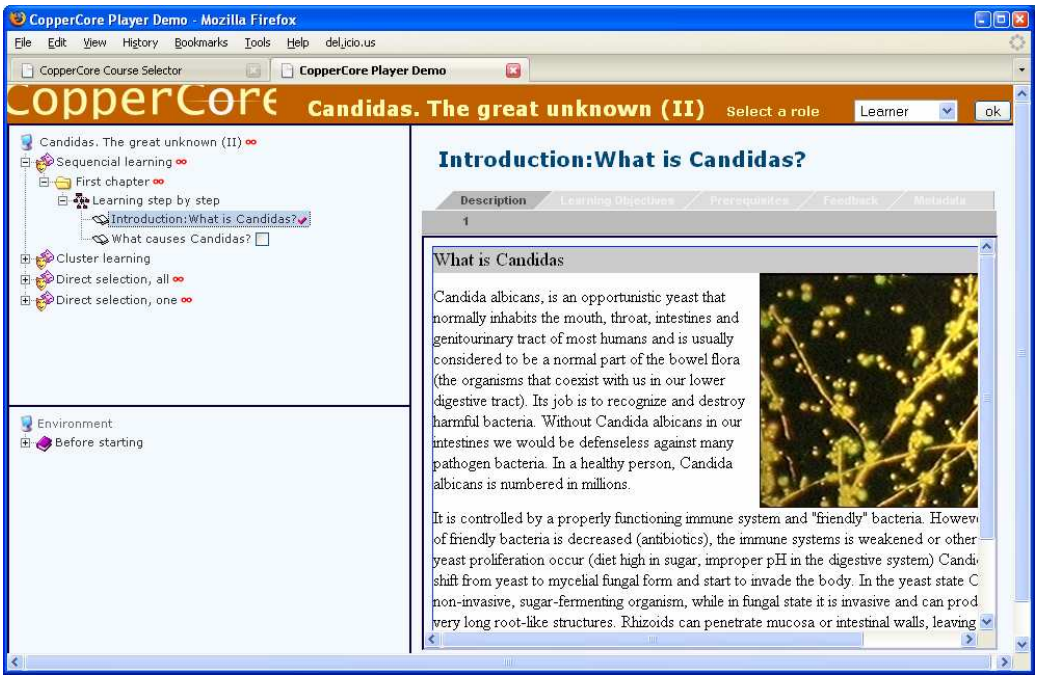


Figure 1: The CopperCore Player

(c) As part of the CopperCore environment, another piece of software designed to be used in conjunction with the Learning Design Engine, is the CopperCore Service Integration framework, (C.C.S.I.) This is a framework designed to allow new services to be added to and extend the Learning Design Framework. These services could be either resources that a Unit of learning could want access to at runtime, such as a chat service or alternatively the service could itself be based on another e-learning specification. One example of this could be IMS QTI.

3.3. Reload SCORM 1.2 Player

The second piece of software that seemed suitable for integration was the Reload SCORM Player (Sharples, P., & Beauvoir, P., 2005). As with all runtime tools developed under the Reload project, the original intention was to have an easy to use desktop application which would encourage users to learn more about the specifications. It would be easy to download and install. Everything would run on the users desktop including the server environment which would be accessed using the local loop (http://localhost). In this respect the tools can be envisaged as single user based and therefore more of a previewing system rather than full blown multi-user system, supporting version 1.2 of the SCORM specification. It comprised of three distinct parts.

a. Tomcat server which the runtime application runs under.

b. Desktop client application used to manage the packages and the web server.

c. Web application which supports the actual playing of the SCORM package within a users browser.

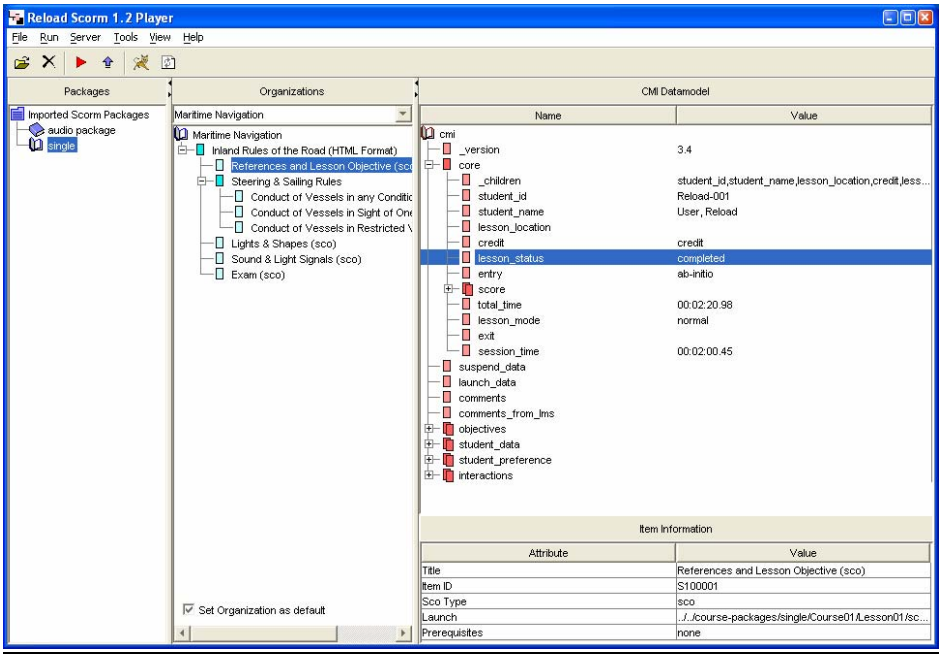


Figure 2: The desktop client application (java swing based)

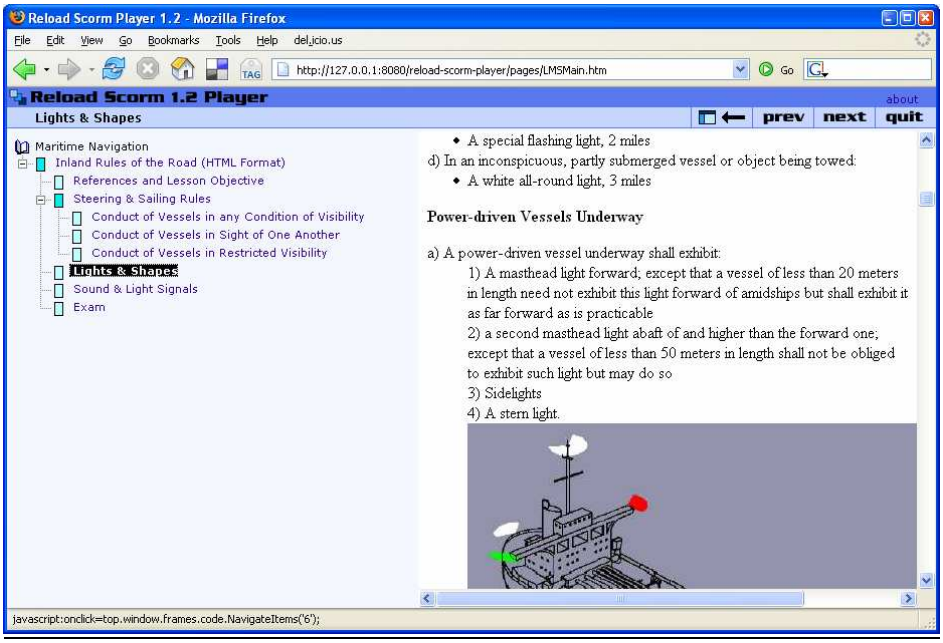


Figure 3: The web application running within a browser (java, jsp and struts based)

The desktop client is used to manage the packages and the web server. There were a few intentions behind having an additional desktop client. One reason was to make the management of tomcat very easy – there are buttons to turn it on & off, as well as the ability to simply click "play", which launches the runtime application in the users browser. Additionally it makes it easy to import, delete and reset packages, or even choose which <organization> within the package to play. It was felt that the ability to see what information was being recorded as part of the process of playing the package, would be a valuable thing to implement. Further to this, the client also provides a monitoring mechanism which will allow the user to see what has happened to each items data model (data model as per the SCORM 1.2 specification)

To summarise, at the beginning of the project we had a standalone single user SCORM 1.2 engine/player available, which sat on the users desktop. Additionally CopperCore was a multi-user Learning Design Engine/Player but not yet able to process any form SCORM related content. The next task was to explore how to effectively combine functionality into one tool.

3.4. Design Considerations

In ADL SCORM terms, there are different types of learning content. Firstly there are "assets". These items are in essence just vanilla web content and do not need to do anything other than to be presented to the user by the LMS, just as any other web page or word document would be. Secondly there are also Sharable Content Objects or (SCOs). A SCO is an item of learning content which is able to interact with the Learning Management System in which it is being played. It must be able to find an adapter which will allow it to do certain things. It must be able to perform certain operations on a defined data model using a defined Application Programming Interface (API). The data model in essence, holds information on how the learner interacted with the system. As an example the SCO could set a value within the data model which records how much time was spent on a given page by a user. The implementation of this differs slightly depending on the version of SCORM being used.

A short paper entitled "How to use IMS Learning Design and SCORM 2004 Together" (Tattersall C, Burgos D, Vogten H, Martens H, Koper R, 2006), proposes one way of incorporating ADL SCORM content with IMS Learning Design. The paper describes how SCOs could be used as items found within a Unit of Learning. These SCOs would be Learning Objects which would appear within the Environment section of a given Activity. The paper expands on this further to describe two ways in which this could be accomplished. As the paper already describes this in detail, here is a brief summary.

The first method involves the SCO being physically located within a separate SCORM aware Learning Management System (LMS). The Learning Object within the Learning Design would simply reference the web address of the SCO running on the remote LMS. The advantage of this is that the Learning Design Runtime Environment does not have to know how to handle the SCOs runtime calls to the API. The main disadvantage is that the Learning Design Environment does not have access to the data model that the SCO interacted with. In essence it works, but there is little or no interaction between the SCO and the executing Learning Design that referenced it.

The second method of using a SCO within a Unit of Learning involves the SCO being physically part of the Learning Design package. It is directly imported into the Learning Design runtime environment. When the SCO needs to be launched within the Learning Design Player, it is passed off to an additional piece of software known as the "Dispatcher". The dispatcher acts as an interface to the Learning Design runtime environment and in this case, the SCORM runtime environment. It is the Dispatchers job to act as the SCORM aware LMS. It should be able to provide the correct environment for the SCO to execute in. The dispatcher will have access to the data model which the SCO interacts with. Subsequently the changes in the SCOs data model could be used to update properties and conditions within the Learning Design. This represents a much better solution as there is true interaction between the SCO and the rest of the Learning Design. The generic "dispatcher" software, had already been written for the CopperCore Runtime Environment and is also known as CopperCore Service Integration framework, (C.C.S.I.).

The second of these two solutions seemed to be the best option. It allows for greater interaction between the SCO and its context within the actual Learning Design. As the dispatcher framework already existed, then the existing SCORM functionality found within the Reload SCORM Player, would need to be adapted for this particular framework. Additionally, some work had already been successfully carried out with integration between CopperCore and IMS QTI. This involved using the "dispatcher" method to enable the learning design player to access a service called APIS, which was able to handle and process the QTI content.

Accordingly, it was envisaged that the SCORM 1.2 engine could be refactored so it can be accessed as a service via CCSI. This would mean that SCORM 1.2 content that may appear inside a Unit of Learning could then be processed during a run. This would be implemented in a similar fashion to how the APIS QTI player was implemented.

The following diagram was drawn to show how the new functionality would fit in with the existing components.

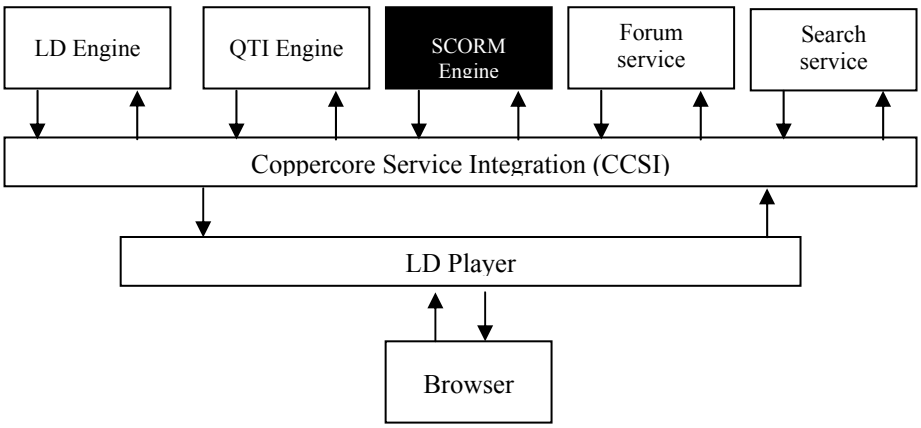


Figure 4: How the SCORM 1.2 Engine fits into the existing framework
Based on the original diagram from http://sled.open.ac.uk/web/tech/ccsi.jsp

3.5. Planning and Design

We had taken the approach of trying to integrate the SCORM functionality, in a similar fashion to how APIS the IMS QTI engine had been implemented within CCSI. However, there are couple of differences in the way SCORM SCOs may be incorporated into a Learning Design, as opposed to IMS QTI content. Whereas IMS QTI content is essentially an XML file which somehow needs to be processed and rendered, a SCO is typically a single HTML page with embedded javascript. This page needs to be able to access an APIAdapter object within the page (or frameset hierarchy) as it is being taken.

The following diagram illustrates the concept of the relationship between the SCO and the API adapter in a typical LMS frameset system (although this is not the only way to do this)…

<table>
<tr><td colspan="2">Top Frame – this contains the APIAdapter object</td></tr>
<tr><td>This frame would typically hold some sort of navigation</td><td>ContentFrame
1. The SCO is loaded into the frameset. typically in this Content Frame
2. It starts to look for the API adapter object within this frame, If not found it will search any parent frames for the object.
3. If it cannot find the adapter, then an alert box will open informing the user</td></tr>
</table>

Figure 5: A typical SCORM frame structure inside the web browser

Note, not all SCORM LMS use framesets. Another involves the use of layers or tags inside a given page. There are a number of ways of actually implementing a SCORM API adapter. Some solutions use a java applet which is embedded into one of the frames, while others may use Active X or another technology.

The adapter needs to be accessible from the SCO using javascript and is referenced as an object. In the case of SCORM 1.2 the object must be declared as "API". Once the SCO has found this object, it can then issue commands to the data model.

3.6. The existing Reload SCORM Player software

The API Adapter

In the Reload SCORM Player, the API Adapter was written purely in javascript. All of the API calls and rules were implemented as a set of objects in javascript. The one drawback to this is that javascript running inside a browser, cannot write to the file system. Thus, there was no way of storing the information between sessions. To solve this problem an additional hidden frame was used within the frameset, which contained a web form. When the data model for the SCO needed to be saved the model was passed as a series of name/value pairs to the hidden frame. The form in the hidden frame was then submitted to the Java servlet backend which could commit the model to disk. The frame in which the SCO resided could not reload itself, because it would lose its current state information. This is why a hidden frame was used.

At the time of writing the Reload SCORM Player code, the technology known as "Ajax" was not widely available. An Ajax call within a web page can call the backend web server for new content without having to reload the entire page.

Reusing other pieces of the Reload SCORM Player framework

As previously mentioned, there are a number of parts to the software. We had already identified that we would need to re-use the javascript API adapter. Additionally we would need to re-use part of the SCORM engine web application. This specific bit of code could handle other attributes of the data model, which we would need to model within the new C.C.S.I. version. Specifically the server side model which was written in java had additional functionality. For example it could hold the state information of a SCO once it has been initialised. It also held the whole model in such as structure that could make it very easy to query – something which would be needed.

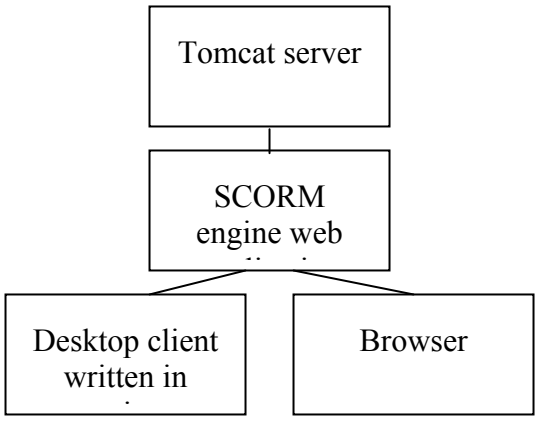


Figure 6: The structure of the SCORM runtime software

One of the first jobs was be to uncouple the constituent parts mentioned above and isolate the engine and user interface parts. The code was written in such a way that this was relatively straight forward. Additionally, we had to identify what exactly the engine as a service should be able to do – what information does it have to process and what information does it need to return in order for it to become an effective SCORM 1.2 SCO service which can be used via CCSI.

At this point we made a design decision. The aim and intention of integrating the new SCO service into CCSI was to be able to execute a SCO and for the results of this to influence Learning Design properties & conditions. If a single SCO is played, then once an appropriate property is updated within the Learning Design, the SCO data model is no longer needed by the Learning Design runtime environment. Thus, the SCO data model could be held in memory until it had been finished by the user, then it could be disposed of. *(Note: a facility to persist the SCORM content in a database was also later added.)*

3.7. Implementation

Changes to the Learning Design Manifest

For a Learning Design runtime environment to be able to process a particular learning object as special SCORM content it must be able to identify it within a IMS Learning Design manifest. There are two types of resource that can normally be found within a manifest. The first is vanilla "webcontent" – meaning any content that can be shown within a browser. The second is "imsldcontent" – a special type that indicates an xml file that needs to be processed and treated differently under the runtime system.

As part of adding in QTI functionality, the authors had decided to extend this list so that a QTI item could be identified by using the type "imsqti_item_xmlv2p0". Following this convention we used “adl_sco_v1.2”

The following xml fragment shows how this appears within the resources section of the manifest.

```
<resource identifier="LO-SCO" type="adl_sco_v1p2"
href="scorm/sco.html">
    <file href="scorm/sco.html"/>
    <file href="api/APIWrapper.js" />
    <file href="api/SCOFunctions.js" />
</resource>
```

The Learning Design runtime environment now has a way to interpret that this resource is a version 1.2 SCORM SCO.

Note: We decided not to use “adl_scorm_v1p2”, as this could be interpreted as a zip package containing a full SCORM 1.2 course, including its own manifest.

Mapping a Learning Design Property to a SCO data model element

One question in marrying IMS Learning Design and ADL SCORM 1.2 together is, “How does the unit of learning interact with the SCO data model? The SCO data model has a certain format of values that can be accessed by the SCO at runtime. The following xml illustrates a typical data model for a SCO, as expressed in XML format and used by the Reload 1.2 SCORM Player.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This is a version 1.2.1 SCORM 1.2 SCO CMI Datamodel-->
<cmi>
  <_version>3.4</_version>
  <core>
<_children>student_id,student_name,lesson_location,credit,lesson_status,entry,sc
ore,total_time,lesson_mode,exit,session_time</_children>
    <student_id>Reload-001</student_id>
    <student_name>User, Reload</student_name>
    <lesson_location />
    <credit>credit</credit>
    <lesson_status>completed</lesson_status>
    <entry>ab-initio</entry>
    <score>
      <_children>raw,min,max</_children>
      <raw />
      <min />
      <max />
```

```
    </score>
    <total_time>00:00:03.30</total_time>
    <lesson_mode>normal</lesson_mode>
    <exit />
    <session_time>00:00:01.38</session_time>
  </core>
  <suspend_data />
  <launch_data />
  <comments />
  <comments_from_lms />
  <objectives>
    <_children>id,score,status</_children>
    <_count>0</_count>
  </objectives>
  <student_data>
    <_children>mastery_score,max_time_allowed,time_limit_action</_children>
    <mastery_score />
    <max_time_allowed />
    <time_limit_action>continue,no message</time_limit_action>
  </student_data>
  <student_preference>
    <_children>audio,language,speed,text</_children>
    <audio>0</audio>
    <language />
    <speed>0</speed>
    <text>0</text>
  </student_preference>
  <interactions>
<_children>id,objectives,time,type,correct_responses,weighting,student_response,
result,latency</_children>
    <_count>0</_count>
  </interactions>
</cmi>
```

As can be seen, there are various elements within the SCO data model that could be advantageous to use within a Unit of Learning. For example the value "cmi.core.session_time" will be used to hold the value of how long a user spent viewing a page.

One method to combine the two, is to design a unit of learning that has IMSLD level properties that are named to correspond with values in the SCO data model. For example,

let us imagine we have a single SCO which we wish to use inside a unit of learning. The SCO is designed to set the "cmi.core.lesson_status" to "completed" once the SCO has been taken. A typical SCO to accomplish this, may look like the following within a browser …

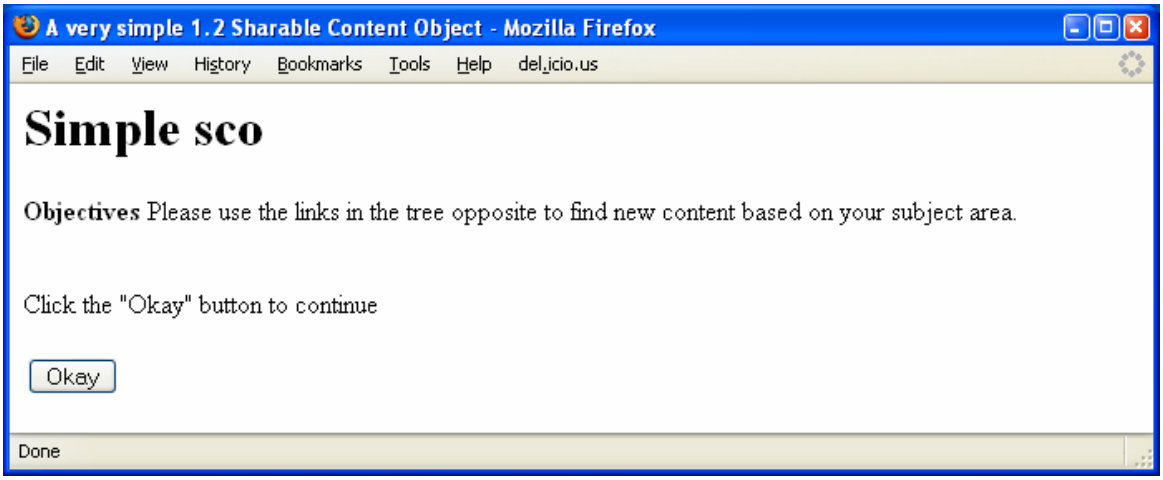


Figure 7: The SCO as seen within a browser

The HTML and javascript used to author this page may look like the following…

```
<html>
<head>
<script language=javascript src="SCOFunctions.js"></script>
<title>A very simple 1.2 Sharable Content Object</title>
</head>
<body onunload="return LMSSetValue('cmi.core.lesson_status','incomplete')">
<h1>Simple sco</h1>
<p>
<b>Objectives</b> Please use the links in the tree opposite to find new content
based on your subject area.
</p>
Click the "Okay" button to continue
<form>
 <table>
  <tr>
 <td><input         type="button"         value="Okay"         onClick         =
"LMSSetValue('cmi.core.lesson_status','completed');LMSFinish('');" name=ok></td>
  </tr>
 </table>
```

```
</form>
</body>
</html>
```

Once the user clicks the button, the “"cmi.core.lesson_status” value is set to “completed” and the SCO is finished.

If the SCO model can set/get this value, we also need a corresponding “"cmi.core.lesson_status” value defined as an IMSLD level B property within the unit of learning. To accomplish this we need to define this property as shown in the following XML fragment…

```
<locpers-property identifier="cmi.core.lesson_status">
        <title>cmi.core.lesson_status</title>
        <datatype datatype="string" />
        <initial-value>not attempted</initial-value>
        <restriction restriction-type="enumeration">not attempted</restriction>
        <restriction restriction-type="enumeration">not completed</restriction>
        <restriction restriction-type="enumeration">completed</restriction>
        <restriction restriction-type="enumeration">passed</restriction>
        <restriction restriction-type="enumeration">failed</restriction>
</locpers-property>
```

Note the use of the enumerated restriction on the values allowed. This corresponds to the values allowed for “cmi.core.lesson_status” in the SCO data model.

At this point we have the value declared within the unit of learning. However, in order for the SCO to influence and change the flow and order of the unit of learning, we would also need to define some sort of condition. The following XML fragment does this…

```
<conditions>
      <title />
      <if>
        <is>
          <property-ref ref="cmi.core.lesson_status" />
          <property-value>completed</property-value>
        </is>
      </if>
      <then>
        <show>
          <learning-activity-ref ref="another-item-that-wasnt-visible-before" />
        </show>
      </then>
</conditions>
```

Here we can see that the condition is set to show a new learning activity if the "cmi.core.lesson_status" property is set to "completed".

We had arrived at the point where we could express using a unit of learning, the SCO itself and the properties we wished to use to influence the outcome of the course.

The final piece of the puzzle for implementation was to create the mechanism for the SCO data model properties to actually reference the same values found with the unit of learning as Level B properties. For this we would need to update the existing Learning Design Player software and then also add some new components

3.8. The Runtime Learning Design Players

The players needed to handle the new SCO 1.2 content. To do this they had to provide the SCO with the correct environment in which to communicate with the API adapter. It had to be then able set/get the correct properties which would subsequently update the unit of learning. The following subsections describe the two main parts of this work.

CopperCore Player

The CopperCore Player is based around a Java 2 Enterprise Edition Servlet. This servlet is responsible for orchestrating the content to be delivered back to the web browser. It is not in the scope of this document to describe how this happens in detail. However, it is important to describe how this process works for SCORM 1.2 SCOs. We will assume that a link to a SCO is visible within a particular environment, but the user has not yet clicked on it…

Figure 8: The student starts the Unit of Learning

The user clicks the tree item which sends a call back to the servlet, with a request for content. The servlet realises that the content is of type “adl_sco_v1p2”. It first creates a link to a new SCO 1.2 service within the CopperCore Service Interface (C.C.S.I.) – See next section. The SCO 1.2 service allows for a new data model to be created for the given SCO. This data model is initialised with certain values, such as the student name for example under “cmi.core.student_name”.

A string containing XML values is then assembled within the servlet which contains this data model. The XML string is associated with a XSLT transformation stylesheet. The servlet returns the XML string and the location of the XSLT back to the browser. The xml string will look similar to the following code…

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xsl/sco12content.xsl"?>
<scorm>
<sco>
<datamodel>
<cmiitem>
   <cminame>cmi.core.student_id</cminame>
    <cmivalue>lduser-001</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.student_name</cminame>
   <cmivalue>LDUser, A</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.lesson_location</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.credit</cminame>
   <cmivalue>credit</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.lesson_status</cminame>
   <cmivalue>not attempted</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.entry</cminame>
   <cmivalue>ab-initio</cmivalue>
</cmiitem>
<cmiitem>
```

```
   <cminame>cmi.core.score.raw</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.score.max</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.score.min</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.total_time</cminame>
   <cmivalue>0000:00:00.00</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.lesson_mode</cminame>
   <cmivalue>normal</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.exit</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.core.session_time</cminame>
   <cmivalue>00:00:00</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.suspend_data</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.launch_data</cminame>
   <cmivalue></cmivalue></cmiitem>
<cmiitem>
   <cminame>cmi.comments</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
```

```
   <cminame>cmi.comments_from_lms</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_data.mastery_score</cminame>
   <cmivalue>0</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_data.max_time_allowed</cminame>
   <cmivalue>10:10:10</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_data.time_limit_action</cminame>
   <cmivalue>continue</cmivalue></cmiitem>
<cmiitem>
   <cminame>cmi.student_preference.audio</cminame>
   <cmivalue>0</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_preference.language</cminame>
   <cmivalue></cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_preference.speed</cminame>
   <cmivalue>0</cmivalue>
</cmiitem>
<cmiitem>
   <cminame>cmi.student_preference.text</cminame>
   <cmivalue>0</cmivalue>
</cmiitem>
</datamodel>
   <url>http://localhost:8080/0/test.htm</url>
</sco>
</scorm>
```

The browser then applies the XSLT stylesheet and the SCO content is loaded into the browser.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:variable name="newline">
<xsl:text>
</xsl:text>
</xsl:variable>

<xsl:template match="/">
<html>
   <!--
  Have to put this style declaration in here because Firefox
  won't render the Iframe properly otherwise.
//-->
   <style type="text/css">
 html,body{
          width:100%;
          height:100%;
          margin:0px;
          padding:0px;
          overflow:hidden;
 }
</style>

<xsl:element name="title">Sco Player</xsl:element>
   <xsl:element name="meta">
      <xsl:attribute name="http-equiv">Content-Type</xsl:attribute>
      <xsl:attribute name="content">text/html; charset=UTF-8</xsl:attribute>
   </xsl:element>

   <!-- include the javascript API Adapter file -->
   <xsl:element name="script" >
      <xsl:attribute name="type">text/javascript</xsl:attribute>
     <xsl:attribute name="src">js/ApiAdapter.js</xsl:attribute>
     <xsl:text disable-output-escaping="yes">&lt;!-- prevent use of empty tag --
></xsl:text>
</xsl:element>

<body><xsl:value-of select="$newline"/> <!-- line break in output file -->
```

```
<script>
<xsl:text                                    disable-output-escaping="yes">function
loadCMIStrings_sco(){</xsl:text>
 <xsl:value-of select="$newline"/> <!-- line break in output file -->
 <xsl:for-each select="/scorm/sco/datamodel/cmiitem">
  <xsl:text disable-output-escaping="yes">loadDataIntoModel("</xsl:text>
          <xsl:value-of select="cminame"/>
          <xsl:text disable-output-escaping="yes">", "</xsl:text>
          <xsl:value-of select="cmivalue"/>
          <xsl:text disable-output-escaping="yes">");</xsl:text>
          <xsl:value-of select="$newline"/> <!-- line break in output file -->
 </xsl:for-each>
<xsl:text               disable-output-escaping="yes">}</xsl:text><xsl:value-of
select="$newline"/> <!-- line break in output file -->
</script><xsl:value-of select="$newline"/> <!-- line break in output file -->

<xsl:element name="iframe">
      <xsl:attribute name="scrolling">auto</xsl:attribute>
     <xsl:attribute name="frameborder">0</xsl:attribute>
     <!-- Also leave the height of the Iframe at 99% - problem on Firefox at
100% -->
     <xsl:attribute
name="style">width:100%;height:99%;margin:0px;padding:0px;</xsl:attribute>
     <xsl:attribute name="id">sco</xsl:attribute>
     <!--
     <xsl:attribute name="src">
   <xsl:value-of select="/scorm/sco/url"/>
     </xsl:attribute>
     -->
</xsl:element>
<xsl:value-of select="$newline"/> <!-- line break in output file -->
<xsl:value-of select="$newline"/> <!-- line break in output file -->

<script>
<!-- this is used to allow the APIAdapter to load before the content -->
var url_sco = "<xsl:value-of select="/scorm/sco/url"/>";
var numberOfAttempts = 5;
findAdapter_sco(numberOfAttempts);
```

```
function findAdapter_sco(num){
if(num>-1){
  try{
   if(typeof API == 'undefined'){

    setTimeout("findAdapter_sco("+--num+")", 300);
   }
   else{
    num = 0;
    loadCMIStrings_sco();
    document.getElementById("sco").src = url_sco;

   }
  }
  catch(ex){
   alert("Error:no adapter found." + ex.message);

   num = 0;
  }
}
}
</script><xsl:value-of select="$newline"/> <!-- line break in output file -->

</body><xsl:value-of select="$newline"/> <!-- line break in output file -->
</html>
</xsl:template>
</xsl:stylesheet>
```

The stylesheet assembles the page in the following way…

1. Includes a reference to the javascript APIAdapter which holds the data model.
2. Creates a layer (of HTML tag) which will hold the actual sco
3. Checks that the APIAdapter has loaded
4. Sets the (source) src attribute of the layer so that it loads the SCO content
5. The SCO loads and is able to access the data model.

The user is then able to use the SCO to set/get values. Once the SCO is finished, or the user navigates away from the page, then it must commit the updated data model back to the server. This is where the javascript APIAdapter needed to be altered.

The SLeD Player

The Sled player essentially works in the same way as the CopperCore Web player. The main difference is that the SLeD player uses the Apache Struts framework instead of using a player servlet to orchestrate delivery of the correct content. In SLeD, a request for SCO content results in a similar set of steps as above. The player will ascertain that a it needs to deal with SCO content. It will create a handle to the SCO web service. (as the CopperCore player does). This will return the XML data as described above. A difference between the two players at this point is that SLeD will render the XML into HTML using a java library and then return the rendered HTML to the browser.

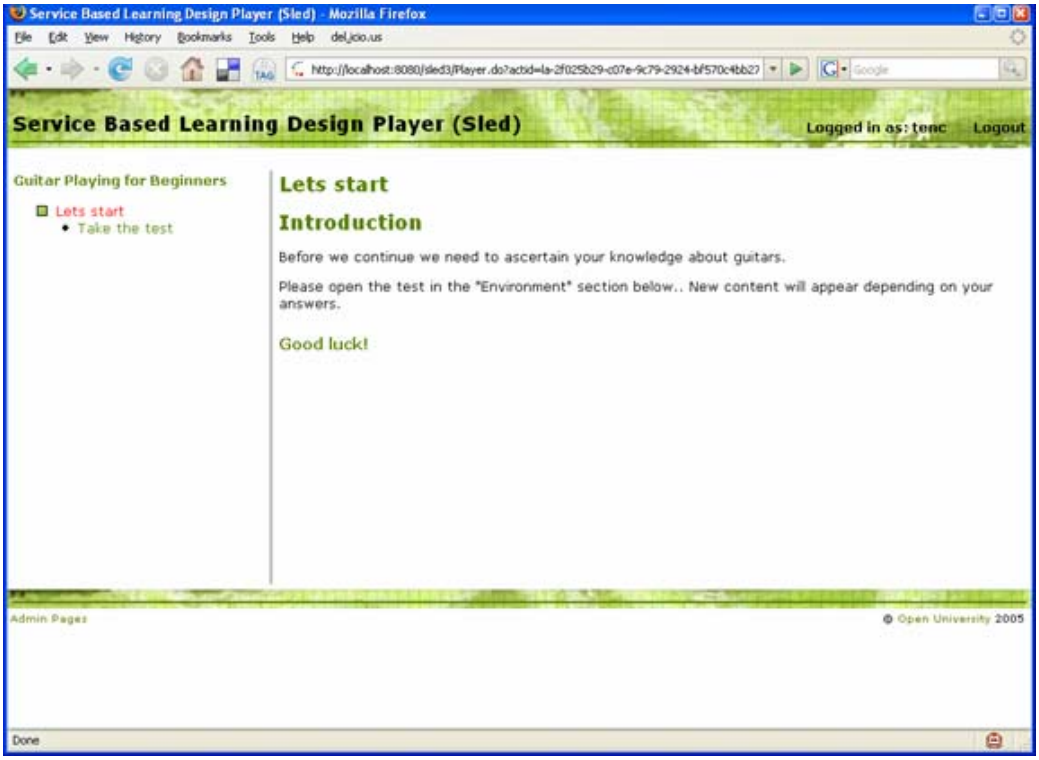


Figure 9: The student starts the Unit of Learning in SLeD

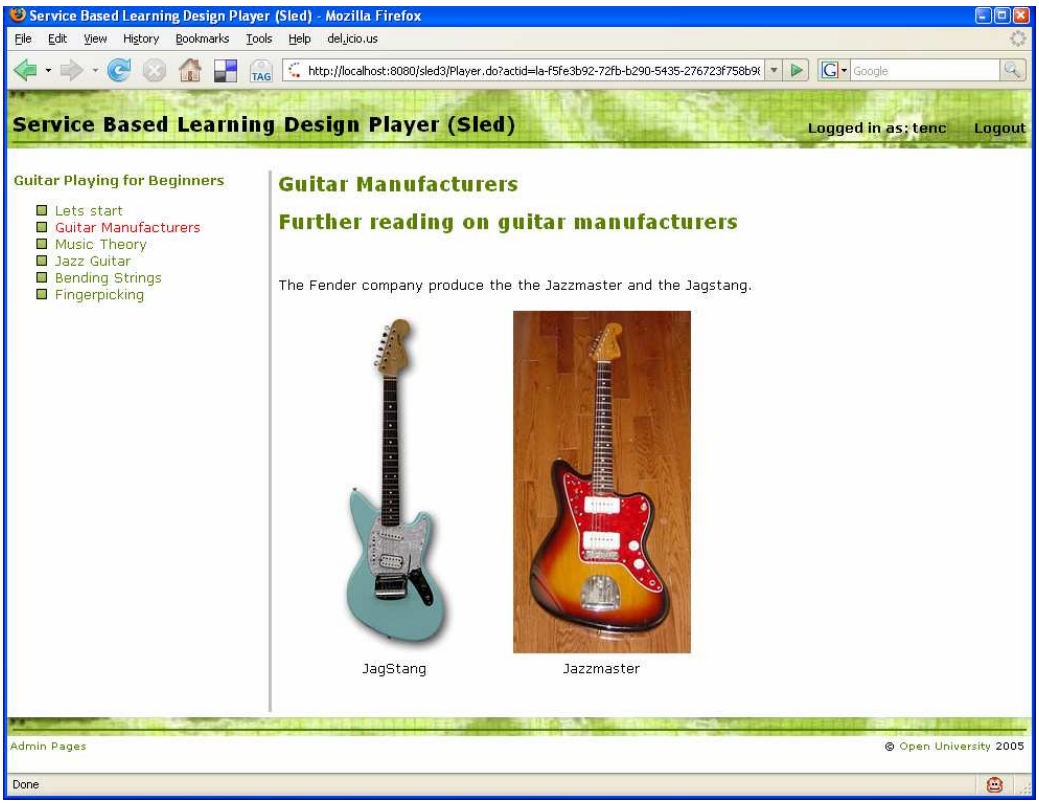


Figure 10: The student sees new content once the SCO has been taken

The Updated API Adapter

The main difference with the new API Adapter as opposed to the original Reload adapter, is that it now uses an Ajax call to submit the data model to the server. What follows is an excerpt of the javascript adapter with an explanation of what it does.

```
function xmlhttpPost(strURL) {                              (4)
    var xmlHttpReq = false;
    var self = this;
    // Mozilla/Safari
    if (window.XMLHttpRequest) {
        self.xmlHttpReq = new XMLHttpRequest();
    }
    // IE
    else if (window.ActiveXObject) {
        self.xmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
    }
    self.xmlHttpReq.open('POST', strURL, true);
    self.xmlHttpReq.setRequestHeader('Content-Type',    'application/x-www-form-
urlencoded');
    self.xmlHttpReq.onreadystatechange = function() {
```

```
        if (self.xmlHttpReq.readyState == 4) {
            updatepage(self.xmlHttpReq.responseText);
        }
    }
    self.xmlHttpReq.send(getquerystring());          (5)
}

function getquerystring() {                          (6)
   // HERE WE ASSEMBLE THE DATA MODEL INTO A QUERYSTRING
   // OF NAME/VALUE PAIRS WHICH THE AJAX CALL CAN PASS BACK TO THE SERVER
   var formData = showCurrentModelState("form");
   var pushTxt = 'pushText=' + escape(formData);
   return pushTxt;
}

function updatepage(str){
   // HERE WE CAN RELOAD NEW SERVER ELEMENTS BACK INTO THE JAVASCRIPT MODEL
}

function updateServer(){                             (3)
   // ****************************
   // HERE WE NEED TO COMMIT THE DATA
   // ****************************
   var strUrl = "webplayer?requestId=6001&amp;type=adl_sco_v1p2";
   xmlhttpPost(strUrl);
}

function LMSCommitMethod(parameter){                 (1)
    // check that this has been called with an empty string...
if (parameter!=""){
this.ServerSco.lastError = "201"
return "false";
}
if (this.ServerSco.isInitialized == "true"){
    setTimeout("updateServer()", 500);               (2)
 this.ServerSco.lastError = "0";
 return "true";
}
else{
```

```
// not initialized
this.ServerSco.lastError = "301";
return "false";
}
}
```

1. The SCO calls LMSCommit(). This means the data model must commit the data back to the server backend.
2. LMSCommit() now calls the method updateServer();
3. updateServer() sets up the URL to which the request is being made and calls xmlhttpPost().
4. xmlhttpPost() - This is the Ajax method. This is responsible for actually submitting the data model back to the server.
5. xmlhttpPost() calls the method getquerystring(). This assembles the data model as name value pairs and then returns the string to…
6. xmlhttpPost**()** which then fires the call back to the server. (the player servlet)

Once the servlet gets this querystring from the ajax call, it then has to submit the data to the piece of software which is able to translate SCO data model elements, into IMSLD level B properties.

This piece of software is the new SCO 1.2 Service and is now part of the C.C.S.I. framework of adapters.

Note: The SLeD player is able to automatically update the structure of the course as the SCO is taken and the Level B properties are changed. However, the CopperCore player is not able to do this. In this case you must refresh the browser to see any structural changes.

3.9. The new SCO 1.2 Service

The SCO 1.2 service has a couple of jobs to do, but its main job is to translate the SCO data model calls into IMSLD properties. As a call is made from the browser

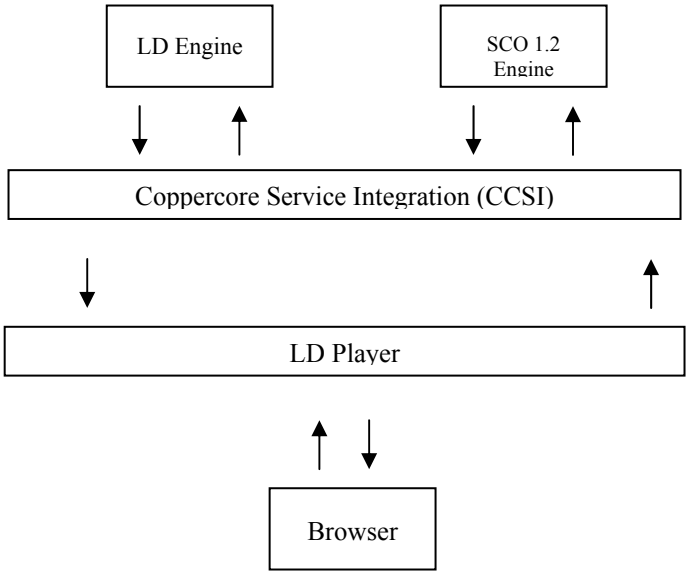


Figure 11: The pieces of the puzzle concerned with SCO communication

The actual SCO 1.2 Service is built as an adapter. This means it is a service built on the C.C.S.I. framework. It also means it has the ability to talk to some of other services that are connected to C.C.S.I. In the case of the SCO 1.2 adapter, it needs to talk to the LDEngine adapter. What follows is a brief explanation of how this is accomplished.

1. The browser submits the javascript data model to the LD Player servlet.
2. The servlet passes the data model onto the SCO 1.2 service. This is possible because C.C.S.I has now been configured to know that there is a SCO 1.2 service available.
3. The SCO 1.2 service parses the name/value pairs it received indirectly from the browser, and formats them in its own data model. It will also update certain elements of the model, such as "cmi.core.total_time". (this means the total time spent on the page)
4. For each SCO data model element it finds, it calls a method which posts events to C.C.S.I
5. The LDEngine service is listening to these event changes in C.C.S.I.
6. For some of the SCO data model elements, the event change will not be recognised. This is because not all of the SCO 1.2 data model elements have been mapped as corresponding IMSLevel B properties. However, for the ones it does recognise, it will receive the change and then propagate that change into the Unit of Learning. Thus, the unit of learning will update and now reflect the changes made at runtime. These changes were instigated by the SCO within the browser.

3.10. The Web Service API

In this section we describe the exposed methods provided by the SCO service to clients (Clients are Learning Design players such as the default CopperCore player and SLeD)

```
public String getValue(String userId, String itemIdentifier,
                                                String propertyName);
```

This method allows a client to retrieve a particular property from the CMI data model. The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning). The propertyName parameter is a name of an element found with the CMI data model. (i.e cmi.core.student_name) . The method should return the value for the given property name.

```
public void setValue(String userId, String itemIdentifier, String
                                propertyName, String propertyValue);
```

This method allows a client to set/update a particular property from the CMI data model. The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning). The propertyName parameter is a name of an element found with the CMI data model. (i.e cmi.core.student_name) . The propertyValue parameter is the actual value of the property to change.

```
public String getXmlModel(String userId, String itemIdentifier);
```

This method allows a client to retrieve the whole CMI data model as an XML string. The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning.)

```
public void pushModel(String userId, String itemIdentifier, int runId,
                                                String cmiData);
```

This method allows a client to submit the whole CMI data model as an XML string to the SCO service.(instead of setting one value – it pushes the whole data model to the service) The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning.) The runId is an integer value passed which denotes on which run within the Learning Design system this SCORM object is being taken. This value is needed by the

SCO service to fire an update event to the Coppercore CCSI module, which in turn, updates the IMS Learning Design Level B properties defined within the Unit of Learning.

```
public String[][] getScoModel(String userId, String itemIdentifier);
```

This method allows a client to obtain the whole CMI data model formatted into a two-dimensional array of name/value pairs. The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning.).

```
public CMIBean[] getScoModelBeans(String userId,
                                  String itemIdentifier);
```

This method allows a client to obtain the whole CMI data model as an array of type CMIBean This bean class has simple get/set values for all name/values of the CMI data model. The client specifies a userId and itemIdentfier (these values specify which SCO model to load – i.e the correct SCO model for the correct item found within the Unit of Learning.).

3.11. Testing the prototype with a real example

Background

The intention behind this unit of learning was to show a real example of the SCO integration with CopperCore. The idea was to author a unit of learning which would contain a SCO which would contain some sort of test. The test would manipulate the SCO data model and set a variety of values for each question, using the "cmi.interactions" section of the SCO data model. The example SCO would be based on the ADL "Maritime Navigation" example. (ADL SCORM 1.2 b, 2001) The SCO test found within the "single course example" would be extracted and changed so that it would be relevant in its new environment. The user would be able to take the SCO test and then based on answers to the test, hide or reveal further learning activities.

What the example does

This is an example which makes use of a Sharable Content Object (SCO) within a Unit of Learning to influence the structure of the course. Six acts are contained with the Unit of Learning. As the user progresses through the course, each acts structure can be updated via properties and conditions. These properties are manipulated via the SCO which has its data model element mapped against the IMSLD properties.

The SCO contains a test with five questions. The Unit of Learning itself contains 6 acts. At the start of the Unit of Learning the user is only able to access the first Learning Activity. From here the user has access to the SCO which is located within the Environment section of the Learning Activity. The user clicks on the link which loads the SCO. The user then tries to answer the questions. Once finished, the user can submit the answers by clicking the button "Post answers". Depending on which questions the user answered correctly, the structure of the Unit of Learning will then change (although a browser refresh may be needed to reflect this). New learning activities should then appear in the following acts. These are designed to give the answers to the questions that the user answered incorrectly and also offer advice.

The user can retake the test again, until eventually he/she will get all of the questions correct. Once the user has done this, the final learning activity located in the final act can be accessed.

Authoring the example

To author this example, the Reload Learning Design Editor (Beauvoir, P. & Sharples, P. 2005) was primarily used. The first few stages are routine for any Learning Design and do not require detailed explanation.

Stage one was to assemble the content and import into the editor. Stage two was to create all of the learning activities, learning objects and environments. Stage three was to assemble the method. This consisted of one play and six acts – each containing a learning activity.

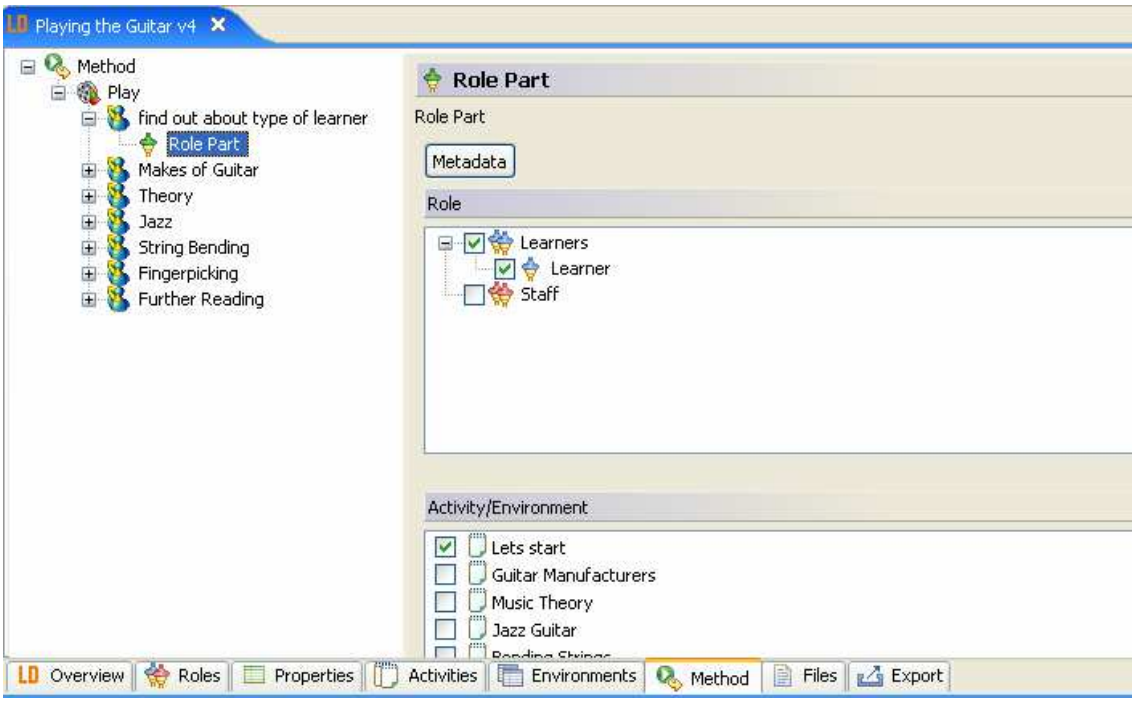


Figure 2: The method of the UOL showing the acts

The next stage was to define the level B properties which would correspond to the SCO data model elements we wished to monitor.

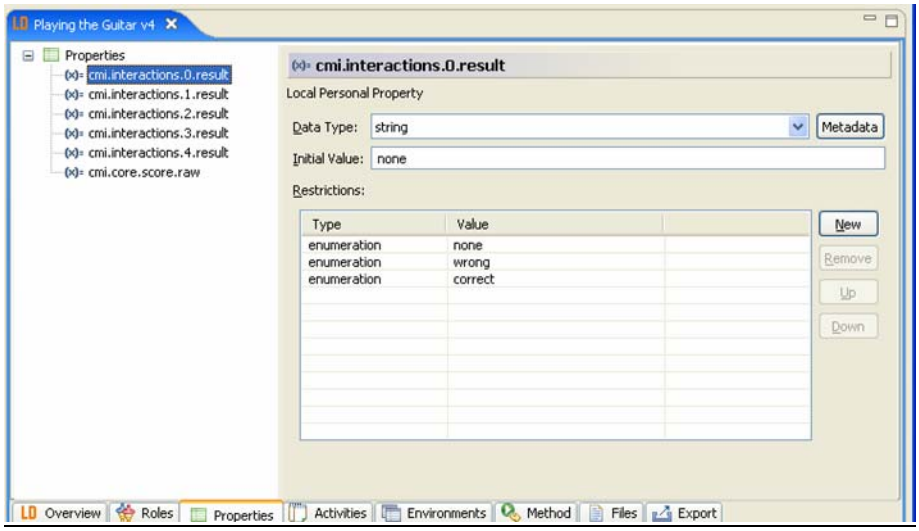


Figure 3: The Properties Pane

Here we can see that we defined six properties.

```
cmi.interactions.0.result
cmi.interactions.1.result
cmi.interactions.2.result
cmi.interactions.3.result
cmi.interactions.4.result
cmi.core.score.raw
```

These are the names of the SCO data model elements that we know the SCO test will manipulate at runtime. We also needed to define some conditions which would change the unit of learning as the properties were updated.

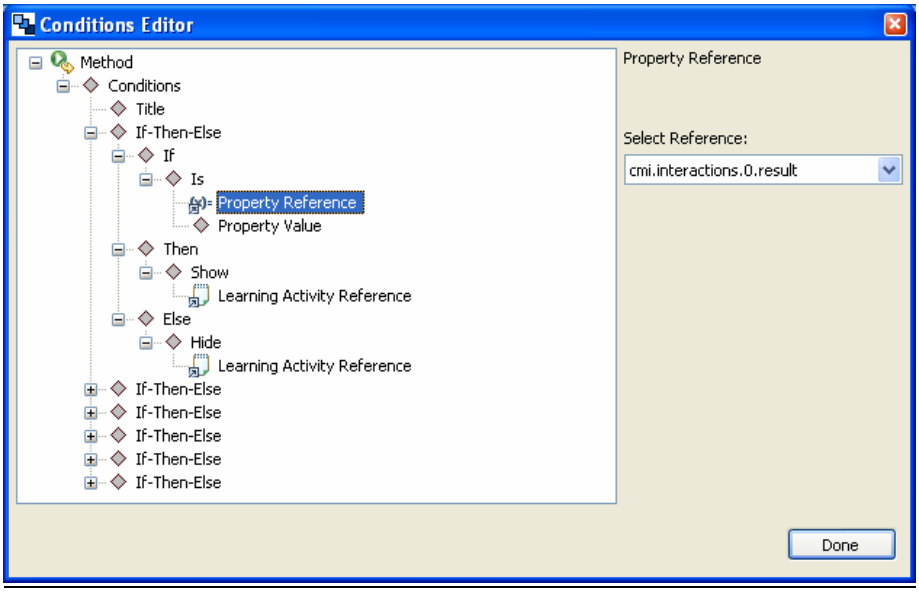


Figure 4: The Condition Editor

In the conditions editor, we define six conditions. The first five conditions essentially are the same and follow this format.

```
If "cmi,interactions.*.result" equals wrong
then show the remedial learning activity for that question
otherwise the question was answered correctly, so hide the remedial learning activity
```

The last condition is slightly different. In this condition, the behaviour was to show a final learning activity once all of the other questions had been answered correctly.

```
If "cmi,core.score.raw" equals 5  (all the questions are correct)
then show the completed/further reading learning activity
otherwise hide the completed/further reading learning activity
```

Once the conditions had been entered, another task was to set the type of content for the SCO test. As discussed earlier, we needed to set this to "adl_sco_v1p2". This does not appear in the drop down list of types for a resource within the editor, but it does allow you to type it in as free text instead.

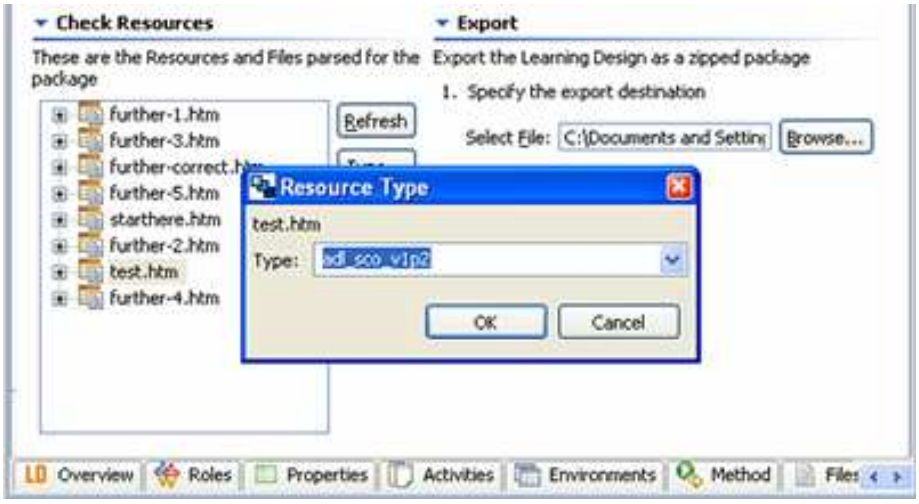


Figure 5: Changing the resource type to adl_sco_v1p2

The package could then be exported. There was one other additional task to finish the authoring process. The zip package was unzipped and the manifest edited by hand. Why? This was because although we had defined our properties in the editor, it had actually (and correctly) set the property title in the XML to the one we entered. However, at runtime, the property identifier also needs to reflect the SCO data model name. The excerpt below shows in bold what needed to be hand edited for each property. (and any other references to it in the manifest)

```
<locpers-property identifier="cmi.interactions.0.result">
        <title>cmi.interactions.0.result</title>
        <datatype datatype="string" />
        <initial-value>none</initial-value>
        <restriction restriction-type="enumeration">none</restriction>
        <restriction restriction-type="enumeration">wrong</restriction>
        <restriction restriction-type="enumeration">correct</restriction>
</locpers-property>
```

Once finished, the example package was re-zipped and then imported into the runtime environment, CopperCore.

Illustrating the example

Figure 6: The Learner starts the Unit of Learning

The course begins. The only item that the user can take is located within the first act. The user clicks the first learning activity here. This tells the user to take a test that is located inside the environment for this learning activity.

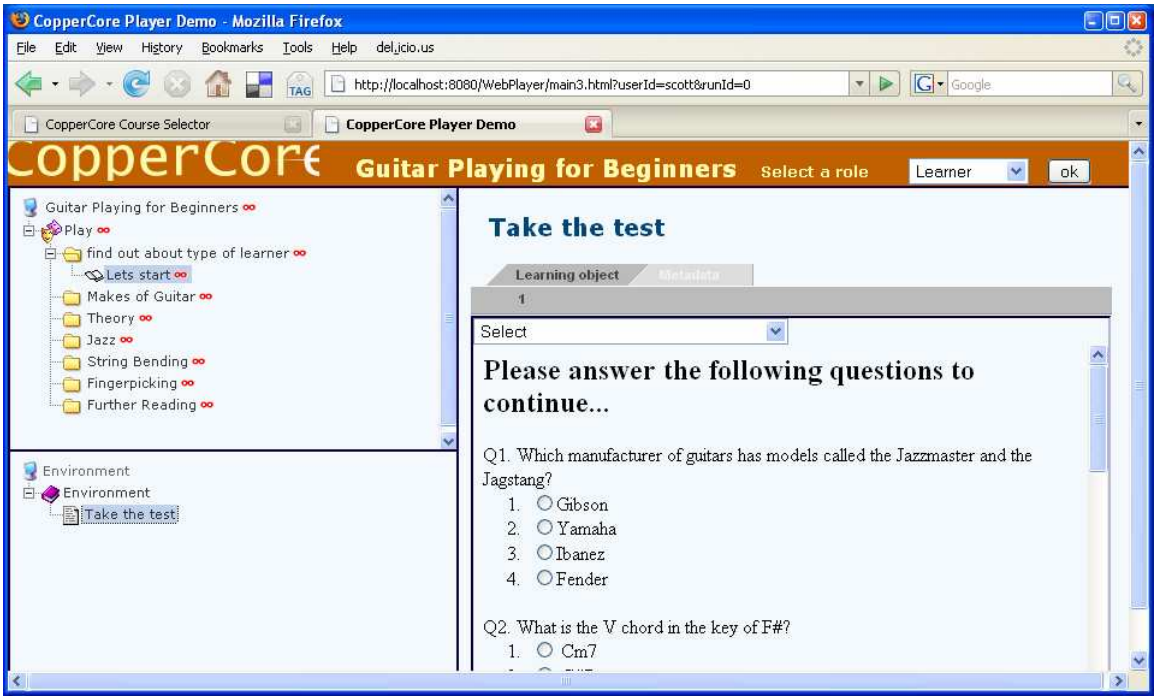


Figure 7: The Learner starts the SCO test

The user clicks on the test. The test is actually a version 1.2 Sharable Content Object. The SCO contains 5 questions. The user can now take the test.

Figure 8: The Learner completes the SCO test and posts results

The user answers the questions then clicks “Post Answers”.

Figure 9: The Learner has answered some question incorrectly

Here we can see that the user got three of the questions incorrect. The questions on “Jazz”, “String Bending” and “Finger Picking” were answered incorrectly. The new Learning Activities that have appeared give the answers and some advice. The user retakes the test in the first act. This time he/she answers the “string bending” question correctly.

Figure 10: The Learner still has not answered all questions correctly

The new course structure only contains two items which were answered incorrectly. The user takes the test. This time we assume the user answers all of the questions correctly.

Figure 11: The Learner has answered all questions correctly

The learning activity located in the final act is now accessible. This informs the user that he/she has answered all of the questions correct and in effect he/she has finished the course.

Limitations of the approach adopted

During the process of designing, implementation and testing of the functionality, we have seen some of the problems we currently face with connecting a new service to an IMSLD runtime system. Complexity is one of the main problems. In the CopperCore Service Interface layer, we have the ability to add in our new service. The problem arises in that actually implementing a new service, requires a knowledge of this specific application programming interface and not a defined and agreed standard. Additionally the framework is quite open, allowing the developer to write his/her own calls between LD Engine and new service. While this allows a large amount of freedom to the developer, it is also very abstract. An additional factor is that integration of services using this approach requires considerable effort by experienced programmers.

3.12. Future possibilities of SCORM Integration

SCORM 2004

SCORM has moved onto version 2004. A future possibility could be that the SCORM service could be extended to support both SCORM 1.2 and SCORM 2004 the main tasks in this work would be updating the API, as it changed between versions. There could be an extended framework within the engine where the service could determine and handle both types of content.

Using the full SCORM package

Additionally further work could involve the use of full ADL SCORM 1.2 (or 2004) packages as learning objects found within an IMS Learning Design package. Processing a full package of SCORM content would seem to be the next logical step in integration.

3.13. Exemplified pedagogical scenarios used in demonstrating the integrated system

Scenario One: Astronomy

The approach adopted makes uses of the role of teacher, and a role for each of the teams, Team A and Team B. The teacher is assumed to assign the Learners to one or other of the teams (to one or other of the roles).

The case study is divided into two Acts. The first act covers the team-based activity of cooperating to understand more about the naming and ordering of the planets, with the teacher offering assistance. This Act is completed when the teacher sees fit. The second Act has an individual activity for the Learners to make the associations, with the teacher monitoring the activity, declaring a winner and completing the unit of learning.

A learning activity entitled “Cooperate to name and order the planets” is defined, together with a learning activity entitled ‘Complete the questionnaire’. Two support activities are defined, “Monitor the Learner collaboration” and “Supervise completion of

the questionnaire". Extensive use is made of Environments containing Learning Objects and Services. The expert interviews are seen as Learning Objects. The forum is an IMSLD Conference of type 'asynchronous' and the chat rooms a Conference of type 'synchronous'. Both the role of Team A and Team B are participants in the forum, as is the Teacher role. In this way all participants in this learning process can make use of the forum. One chat room is associated with each of the teams so that only intra-team communication is possible. In the worked out scenario, the teacher has not been granted participant or observer rights so that the chat is essentially private to a team (this could be modified so that the teacher is afforded a window on the interaction).

Two Activity Structures are defined to reflect the different situations of Team A and Team B. Each contains a reference to the learning activity of "Cooperate to name and order the planets", and to the environment containing the shared forum service. In addition the Activity Structure for Team A has a link to an environment containing Team A's Expert Interview and Team A's chat room. Similarly, the Activity Structure for Team B has a link to an environment containing Team B's Expert Interview and Team B's chat room. In this way the cooperation and competition is facilitated. In addition to participating in the forum, the teacher is given the opportunity to set a property indicating that the first Act should end. Once set, the flow of the process moves onto the second act where each user provides an answer (via an IMSLD locpers-property) to the ordering and naming question. The teacher is provided with a view on these answers (via the monitor service) together with a mechanism to end the process and declare the winner (via a feedback-description shown on completion of the second act and containing the value of a property through global elements in so-called imsldcontent).

The setting of properties by the teacher is supported in the current version of CopperCore, with the user interface control being generated from the type of the property (e.g. Boolean leads to combobox). The monitor service, through which the teacher is able to follow the Learners' attempts at the questionnaire, is implemented within the player which accompanies the CopperCore engine. Further service integration into CopperCore-based environments has been the topic of recent R&D [4] and a loose level of integration has been achieved with Moodle. Through this integration, Moodle's forum services are used to facilitate the inter-team cooperation, including the teacher participation.

At the time of writing, no chat service has been integrated with the CopperCore Service Integration layer, although the TENCompetence project (www.tencompetence .org) will seek to carry out integration of Jabber during the project.

Opportunities for observation or monitoring have been incorporated into the design. First, since the teacher is also a participant in the forum, s/he is able to observe events. Had the choice been taken to offer the teacher insight into the chat rooms, this could have been modelled either by making the teacher a participant, or an observer. Further observational facilities are provided by the use of IMS LD's monitor service when linked to specific properties (e.g. responses to questions) for particular roles.

In terms of the way in which observations can be used to modify the activity's progress, possibilities can be included in the design to have activities, acts, etc be completed when a value is set. This can be as simple as having a flag be raised when a

member of a particular role sees fit (as illustrated in this example), through to more complex conditions in which average scores or numbers of users completing can trigger further events

Activity Diagram

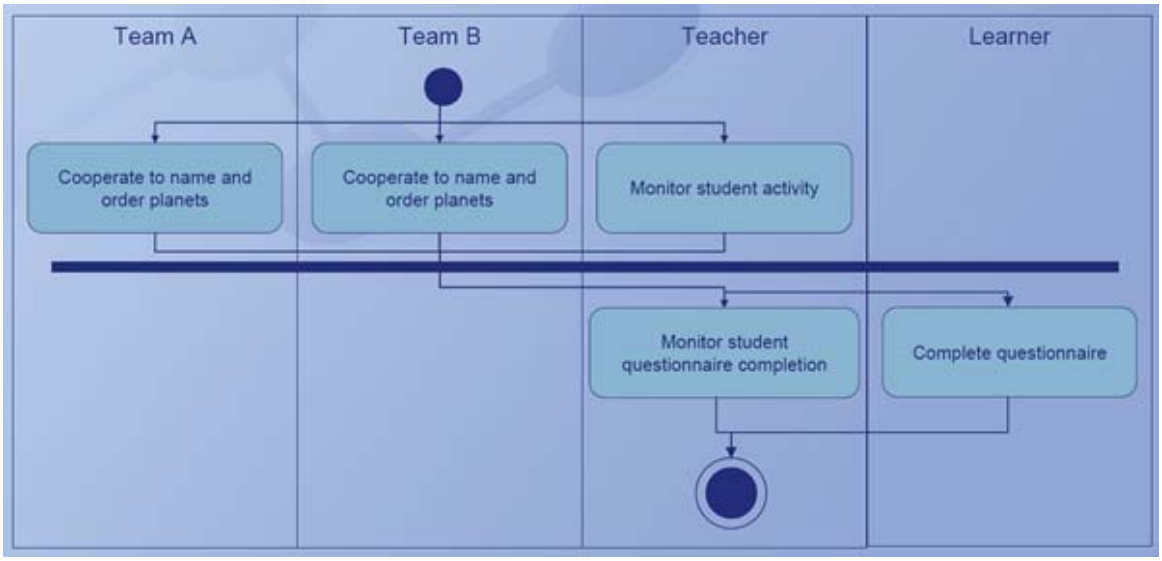


Figure 12: Activity diagram for "Astronomy" Scenario

Scenario Two: Guitar Playing for Beginners

A guitar teacher is searching through a repository, or alternatively is searching the internet looking for reusable content, concerning "guitar playing" aimed at an introductory level. . S/he is looking for content that would be suitable for a self test type scenario for someone with no or little experience with guitar playing. The outcomes from the test could advise the Learner on issues related to that specific topic. The teacher finds a SCORM 1.2 package which contains various resources that s/he would like to reuse. In particular is a small five question test that seems ideal as a simple way to identify the Learner's specific area where s/he needs guidance. The teacher wants to reuse the SCORM based test, but also tailor the questions and add different content. The teacher also wishes to model this pedagogical scenario using IMS Learning Design.

Firstly the teacher downloads the SCORM .12 archive file and unzips it. The teacher then identifies which part of the package contains the test which s/he wishes to reuse. The teacher then extracts this file along with any other files that are relevant to it. (e.g. Javascript files) The teacher edits the test or (Sharable Content Object) in a HTML editor and changes the questions to suit the new intended context for the test.

Next the teacher thinks about which roles would be needed during this course. The course will be modelled on a single user, the "learner". The learner will progress though the Unit of Learning by him/herself. The Unit of Learning will contain seven acts. The first act serves as the placeholder for the SCORM test. A learning Activity will reside in

the first act. It will instruct the user to take a test located in the Environment section. The test will have five questions, which will be able to update certain IMSLD level B properties once submitted. The properties will have conditions attached to them. Changes in conditions will allow certain acts to either “hide” or “show” feedback content, based on whether the Learner answered the question incorrectly. The user will be able to retake the test in the first act, over and over again until s/he gets all of the questions correct. Once all of the questions have been answered correctly, all other feedback will be hidden. The final act will display a new Learning Activity which will inform the Learner that all of the questions have been answered correctly and where to find further reading materials.

On the following page is the activity diagram for “Guitar Playing for Beginners”.

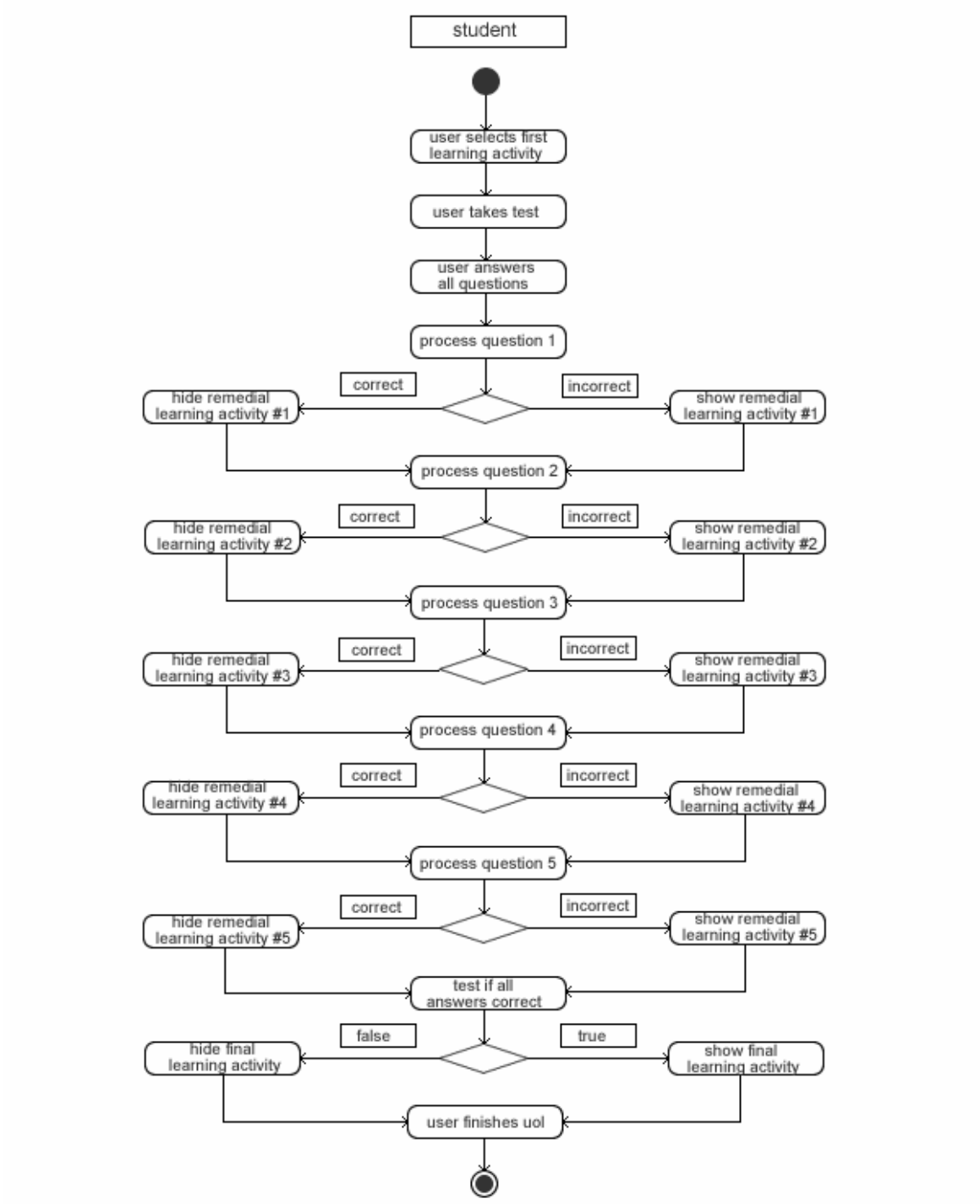


Figure 13: Activity diagram for "Guitar Playing for Beginners" Scenario

3.14. Conclusions

The system described in this section provides an effective and tested solution to the integration of the two principal specifications for learning objects and activities: SCORM and IMS LD. This is in itself a new and valuable contribution to the interoperability of learning activities.

However, the goal in this task is to develop a generalisable solution. The process of designing, implementation and testing of the SCORM integration clarified the problems and complexity posed by connecting a new service to an IMSLD runtime system. The CopperCore Service Interface layer provides the necessary framework, but actually implementing a new service requires knowledge of this specific API, and not a defined and agreed standard. The framework also is quite open, allowing the developer to write his/her own calls between LD Engine and new service. While this allows a large amount of freedom to the developer, it is also very abstract.

The effort involved in integrating SCORM and IMS LD suggests that it is not a solution which can meet the need for the agile integration of a large number of services into IMS LD, which is required for the TENCompetence Communication Protocol. This has led to the development of a new line of research and development, based on the use of multi-user widgets, which we describe in the following two sections of this Annex.

4. Improving and updating IMS QTI Runtime

As described in detail in Annex 3, the assessment strategy of the TENCompetence project requires IMS QTI runtime. As there is no runtime implementation of the latest version of the specification (v2.1) there was a need for the project to fill this gap. This section describes that work, and it consists of a paper presented at the TENCompetence Manchester Open Workshop.

A QTI Management System Implemented for Service Oriented Architectures

Josep Blat, Toni Navarrete, Ayman Moghnieh and Helena Batlle Delgado

Departament de Tecnologia. Universitat Pompeu Fabra, Passeig de Circumval.lació, 8. 08003 Barcelona, Spain
E-mail: {josep.blat, toni.navarrete, ayman.moghnieh, helena.batlle}@upf.edu

Abstract: The IMS Question & Test Interoperability specification describes a model for the representation of questions and tests data and their corresponding results reports. In this paper, we discuss the implementation of a modular-structured online engine for the (QTI) specification, aiming at providing a framework for the development of user friendly IMS QTI applications. This work is considered as a continuation of the APIS project that intends to develop an infrastructure that sustains interoperability and web servicing of IMS QTI.

Keywords: eLearning, IMS Learning Design, IMS Question & Test Interoperability, Usability, Assessment.

4.1. Introduction

The ability to remotely evaluate the acquisition of knowledge remains vital for formal pedagogical models within the e-learning framework. This ability is also usually needed in the case of non-formal learning approaches. A specification that tries to solve the lack of interoperable evaluation of knowledge is IMS Question and Test Interoperability (QTI). IMS QTI can be used with other specifications, namely with IMS Learning Design, which was conceived as a technical specification that allows the modelling of processes, resources, and services used in any learning environment.

The existence of e-Learning interoperability specifications (such as IMS QTI and IMS LD) is necessary for technology convergence, supporting the coexistence of different products. Standardisation applied to learning technologies mainly consists of facilitating interoperability (technological compatibility), reusability and durability. It also opens a new world full of pedagogical possibilities.

Despite the important role that the IMS Question and Test Interoperability has, end-users such as teachers and tutors find it extremely impractical to work with it unless they have a clear, well-documented, open source implementation. In general, the lack of such implementation hinders the communal adoption of specifications and their evolution towards standardisation. Hence, our work aims to provide such an implementation, taking into consideration the usability requirements and potential service design associated with the IMS QTI specification. The resulting implementation comprises a QTIv2.1 compliant engine, as well as a simple player compatible with this specification version. It also supports a large number of QTIv2.0 elements, and some other useful features. Furthermore, the compatibility with service oriented architectures (SOA) is guaranteed due to its modular service oriented implementation.

This paper is structured as follows. Section 2 discusses the state of research and development related to IMS QTI. This includes projects as APIS (Barr et al, 2006), which had previously started working in this field, as well as the motivations that have driven us to select APIS as the basis of our work. Section 3 presents the improvements and implementations carried out, followed by Section 4 which discusses the testing techniques used. Finally, Section 5 summarizes the main contributions of our work and briefly describes some lines for future work.

4.2. Previous works on IMS QTI

The IMS QTI specification enables the exchange of question and test items between authoring tools, item banks, test composition tools, learning systems, and assessment delivery systems to name a few. It principally describes a data model for the representation of questions and tests and their corresponding result reports that makes the interchange of information between systems possible. The model is defined in a standard eXtensible Markup Language (XML) format. The QTI specification remains extensible and customisable to allow specialized or proprietary systems to easily adopt it.

A good indicator of the significance of QTI is the increasing number of projects relying on this specification. Some examples include: TOIA (Tools for Online Interoperable Assessment), Sled (Service Based Learning Design Player), R2Q2 (Rendering and Response processing services for QTIv2 questions), FREMA (eLearning Framework Reference Model for Assessment), APIS (Assessment Provision through Interoperable Segments). The IMS QTI world is evolving, and continuously improving its functions by trial and error adaptation.

The basic elements of the QTI specification are:

Item: is the smallest interchangeable QTI element that stores the question presented to the user along with the associated metadata such as the reproduction instructions, user answers processing mode, hints, and feedback.

Section: represents a composite part of the assessment test or exam.

Test: is an entire QTI instance that embodies a single assessment test. Its structure is divided into sections and subsections and contains sequential

information along with the method(s) to use for combining individual questions scores/marks to form the overall test grade.

QTI v2.0 focuses on simplifying the concept of individual question stored in the item QTI element, while aspects such as the grouping of questions in sections or exams are left out. QTI v2.0 introduces changes in interactions and new variables types, which, in turn, induce modifications in the processing of question responses. On the other hand, the formatting of content text was included in the specification and based in XHTML. QTI v2.0 also introduced new kinds of templates along with feedback, hints, and other additions. Another valuable addendum is a method for integrating QTI and IMS-LD through shared access to the variables generated by users' interactions with the QTI player.

QTI v2.1, the latest release at the time of writing this paper, completes the update from version 1.x to 2.x by adding missing QTI elements such as those associated with the grouping of items in sections or tests, alongside a new results report. To simplify the implementation of this extensive and complex specification, we searched for a good system to upgrade. But, despite community enthusiasm, no reference implementation of QTI v2 had been done. In this context, two relevant projects are related to our needs: APIS and R2Q2 (already named above).

Assessment Provision through Interoperable Segments (APIS) engine (Barr et al, 2006), was originally created by Strathclyde University (in May 2004) to increase interoperability of the IMS QTI specification in practice by providing a liberally licensed open source implementation of the non-controversial elements of the specification (IMS QTI, 2006). They planned to implement a modular item-rendering engine in line with the version 2.0 of the specification. This engine addresses the operations required by potential tools defined in the Open Knowledge Initiative (OKI) and IMS Web Services.

Rendering and Response processing services for QTI v2.0 questions project (R2Q2), which started in March 2006, aims to build a new implementation from scratch, taking into account the previous outputs from other projects, and learning from them what not to do. They needed to implement QTI v2.0 from a completely new code base since their objective was to provide a definitive rendering and response processing engine, properly structured, for this version. Due to its functional-modular design (Renderer, Processor) and use of internal Web Services, the system facilitates future enhancement and can be changed to suit any application.

Finally, we chose to extend APIS (Barr et al, 2006), a basic implementation, as by the time our project began, R2Q2 had not started, and APIS was already finished. Several new upgrades have been implemented in the APIS engine UPF-version, mainly referring to: test context instead of just items (questions), a wide range of new elements test-related, new and more complex response processing and new types of interactions. If some expansion of the work realized was to be done, it could be implemented without problems. We hope that the work builds on APIS and apart from contributing to the e-learning framework with an implementation of a quite complex QTI engine, will also be helpful for any other QTI v2.0 engine to be upgraded to QTI version 2.1.

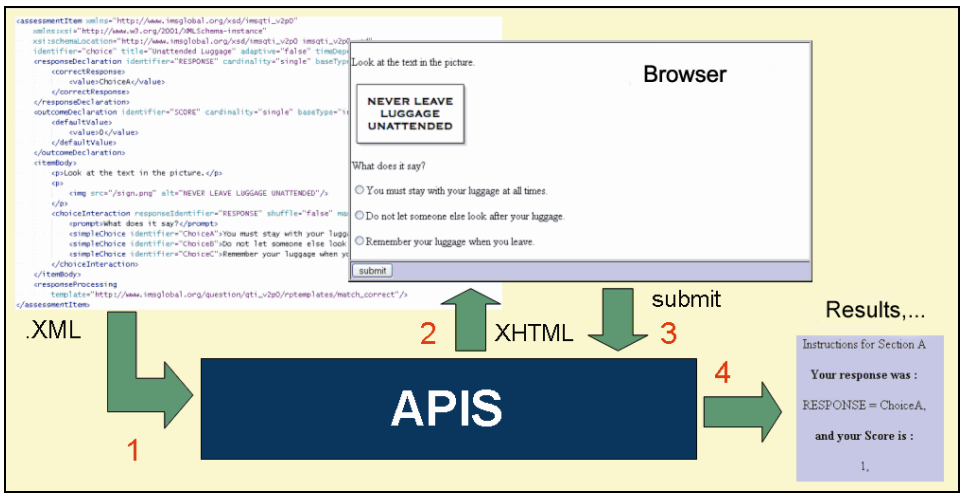


Figure 1: APIS working flow

The basic working flow of the APIS engine is shown in the above figure [fig1].

In order to evaluate an XML document for its QTI compliance, it is loaded into the engine for processing (1). APIS reads, processes, and validates it. Within this first step, the system will return an error message if the document is not well formed or will continue to the next flow stage. For well-formed QTI compliant XML, APIS will use a simple interface integrated (1) into the engine in order to show QTI data to the user and allow interaction (2). This way, the user can answer the displayed question(s) (3). The submitted answers will be processed by APIS following the instructions given in the same XML document. Finally, some kind of feedback is returned to the user, such as score, hints, and more.

4.3. Upgrading the APIS engine

Although a number of QTI-IMS specification implementations are currently available, none of these implementations is open source; in other words, it is not possible for the communities of developers, especially small organisations with limited resources, to take full advantage of existing assessment tools. Our work focuses on improving the implementation of the APIS engine described above to make it compliant with the specification's latest version (v2.1). The steps followed and changes made to obtain a final engine, which would support test process and new added characteristics, are addressed below.

We started by expanding the previous QTI version 2.0 implementation to reach completeness since the APIS state of the art version does not address all the elements in the specification. In particular, APIS does not implement text-based and graphic interactions, new data types and associated cardinalities, logical and other types of expressions, and the upgraded processing of answers and responses. These functionalities enable the management of tests that require the writing of different amounts of text or some graphical interaction.

The upgrade resulted in an engine with almost all interactions implemented, except for: the graphical ones, some of the miscellaneous ones and few of the text related ones. All the data types, apart from files, were now included, also cardinalities and most of the expressions. In addition, a more complex response processing was implemented. Furthermore, the newly added data types enabled the formulation of questions whose answers could be composite such as when test takers are required to select several answers from different pools as a response to a test problem.

Our work proceeded to change the entire engine structure to convert it to QTI v2.1. As this version accepts not just questions but entire tests as well, we have introduced critical changes to the principal components of APIS including the main engine class. Processing of test units thus has become more hierarchical, where tests are considered both as units and groups of divisions. QTI v2.1 appends concrete characteristics for the test's divisions including time constraints, navigation flow style, and others. These paradigms and definitions were implemented and added to the improved engine. This required the adaptation of some of the previously existing processes and elements.

The resulting QTI engine is now available on-line at Sourceforge: http://sourceforge.net/projects/newapis

4.4. Testing and evaluation of the developed version

Although the QTI specification has not yet been completely coded, the resulting engine is quite robust, as preliminary testing and evaluation confirm. To demonstrate that the resulting engine works correctly, we based our implementation on a set of examples provided by IMS Global Learning Consortium (the IMS QTI specification creator). We have adopted these examples as our benchmark for testing and evaluating our implementation since they were published to address and illustrate the specification and tackle it with a wide range of approaches to cover several levels of complexity. The examples are provided in XML format in compliance with QTI requirements. APIS can validate tests in XML format by visualizing their content or returning an error message whenever a file is detected to be non-compliant with the specification. Furthermore, once the test is validated and displayed in the browser, when the user interacts and submits responses, APIS can also process whether the answer is correct or not. The only condition for this functionality to be permitted is that some correction instructions must be properly defined within the XML. By passing these example XML files to the upgraded APIS engine, we can prove the proper working of the upgrades.

The actual state of the engine, referring to QTI version 2.1, is represented in the following figure [fig3]. There, all the elements included in this version of the specification are represented by different boxes. The colours show what has been proved to be implemented and working correctly -green; elements to be done or not yet finished -orange; and finally we can observe where QTIv2 would be placed -yellow.

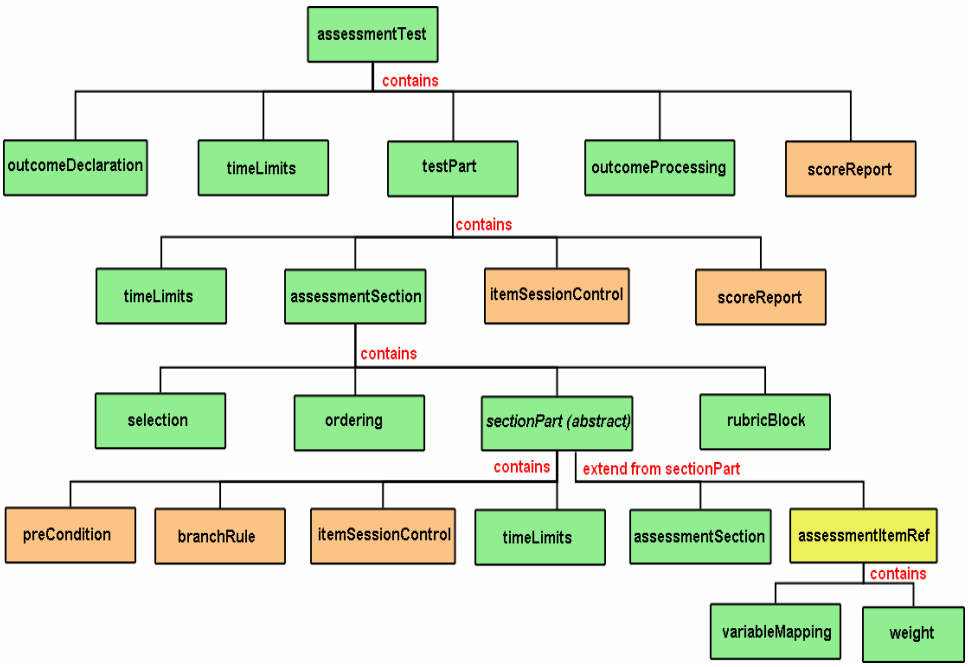


Figure 2: QTI v2.1 scheme of elements

4.5. Application

The new QTI v2.1 compliant engine can be used to build several useful tools that implement IMS specifications. These tools using the APIS engine such as editors and players would be included in the framework. In the following figure [fig 3] we represent the situation of the APIS engine in the application architecture. As users only interact with the framework, the engine remains invisible. Hence, the engine represents the interface of the specification and ensures interoperability and the exchange of information between the tools. The engine's role permits the development of user-friendly tools where usability becomes central. The service orientation characteristic of APIS supports its easy addition to the service-oriented architectures (SOA, 2006). An example of this kind of architecture is SLeD (Service Based Learning Design Player), a Learning Design Player which can easily take advantage of the APIS engine and integrate its assessment properties into its executed learning processes. We are testing and experimenting with the potential of such compatibility, and we have developed several different scenarios in Learning Design for that purpose.

Previous work on a Question & Authoring editor called QAed has been carried out at Universitat Pompeu Fabra. QAed offers a set of basic services related to questions and assessment authoring such as create, edit and search. It also presents a set of extra services centred in the practice of assessment and questions authoring like work repository, assessment categories, basket cart, and HTML preview. An important added value is content reusability, which is achieved through import and export of both assessments and questions in XML files. It enables the interoperability with others systems, like players using the IMS QTILite standard as exchange format (Sayago et al,

2004). The possibility of a new tool that meets both usability requirements, based on QAed's experience, and QTI 2.1 seems interesting.

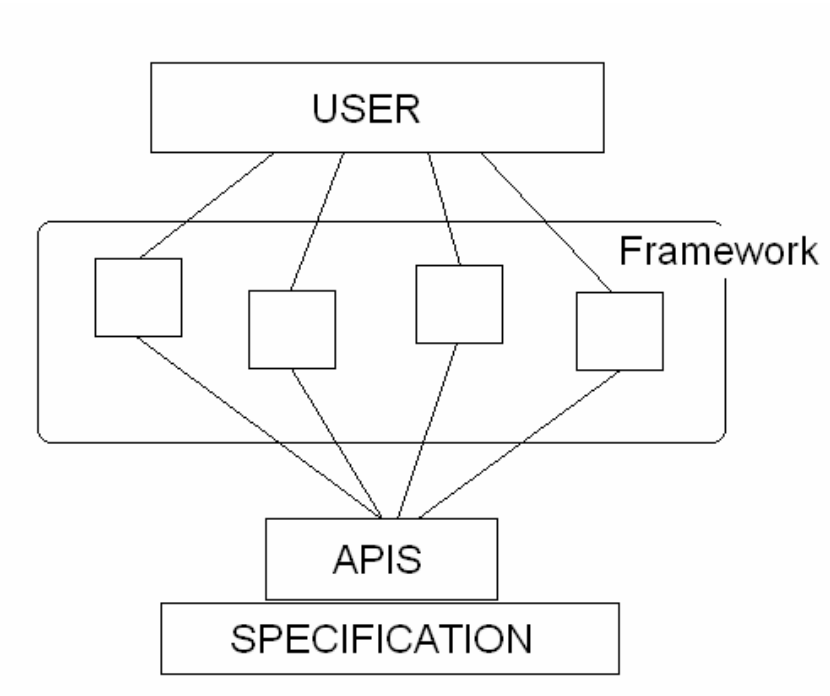


Figure 3: APIS's location between framework and specification

4.6. Conclusions and future work

Assessment plays a significant role in the learning process, and consequently there is a clear need for IMS QTI-compliant engines that make it possible to run interoperable assessment tests. This has been the aim of the upgraded APIS engine, which supports the execution of complex tests with a variety of types of questions, as well as the processing of the results.

It is important to remark that without open-source, well documented implementations of IMS specifications, their adoption by the potential user-base stays extremely slow. We have invested our efforts to upgrade the APIS implementation of the IMS QTI specification since such libraries are open-source and well documented.

Furthermore, assessment is not an isolated element in learning and should be integrated in the workflow of learning activities. The upgraded APIS engine is currently being integrated through services within an experimental platform for running LD Units of Learning. We are currently developing this platform, based on SLeD, where different LD services and tools are being integrated within a single framework. Upon completion, APIS will be integrated within this framework where pilot Learning Design courses will be serviced. This will enable the integration of assessment in the workflow of LD activities, in particular, determining different learning paths according to assessments results.

Acknowledgment

This work has been partially sponsored by the TENCompetence Integrated Project that is funded by the European Commission's 6th Framework Programme, priority IST/Technology Enhanced Learning. Contract 027087 (http://www.tencompetence.org)

References

Bacon, R.A (2006) A review of the IMS Question & Test Interoperability: Item version 2.0, Last Access: November 2006, http://ford.ces.strath.ac.uk/QTI_documents/v2review.

Barr, N. and Sclater, N. (2006) Assessment Provision through Interoperable Segments, Last Access: November 2006, http://ford.ces.strath.ac.uk/APIS.

Masie, (2006) Making sense of Learning Specifications & Standards, Last Access: November 2006, http://www.masie.com/standards/s3_2nd_edition.pdf.

Sayago, S. Martinez, J. Garcia, R. Blat, J. Griffiths, D. and Casado, F (2004) Design and evaluation of a simple eLearning authoring tool, in Navarro, R. and Lorés, J. eds. HCI related papers of Interacción 2004. Springer, 2006, pp: 81-89.On-line at: http://www.iua.upf.es/~dgriffit/papers/sayago-griffiths_lleida.pdf.

Websites

FREMA project (2006) eLEarning Framework Reference Model for Assessment, Last Access: May 2006, http://www.frema.ecs.soton.ac.uk.

IMS Global Learning Consortium (2006) eLEarning Framework Reference Model for Assessment}, Last Access: November 2006, http://www.imsglobal.org.

IMS QTI (2006) Question & Test interoperability documentation v2.0/v2.1, Last Access: November 2006, http://www.imsglobal.question/index.html.

IMS LD (2006) Learning Design documentation, Last Access: November 2006, http://www.imsglobal/learningdesign/index.html.

IMS GWS (2006) Web Services documentation, Last Access: September 2006, http://www.imsglobal.org/gws/index.html.

JISC (2006) JISC website, Last Access: September 2006, http://www.jisc.ac.uk.

OKI (2006) OKI project website, Last Access: November 2006, http://www.okiproject.org/.

R2Q2 (2006) Rendering and Response processing services for QTIv2 questions, Last Access: July 2006, http://www.r2q2.ecs.soton.ac.uk/.

SLeD (2006) SLeD project webpage, Last Access: November 2006, http://sled.open.ac.uk/web.

SOA (2006) Web services and Service-Oriented architectures, November 2006, http://www.service-architecture.com.

TOIA (2006) Tools for Online Interoperable Assessment, June 2006, http://www.toia.ac.uk.

5. The TENCompetence Connection Protocol

5.1. Approaches to implementing a connection protocol

In considering a connection protocol it is worth considering Bill Olivier's input to the WP6 White Paper, which provides some valuable pointers in this respect (Koper, R; Tattersall, C (Eds.) 2005):

> *A new 'open service' element is proposed in which the service type and role elements are also open. This would enable any service type, and its specific sets of roles/ permissions, to be supported. While this would provide great flexibility and allow for the evolution and use of new services, the downside of this would be to weaken interoperability:*
>
> - *On set up, systems may not recognise the service and hence would not be able to automatically select and instantiate it appropriately*
> - *When a large number of services have been created, no individual site would be able to support them all, and hence Units of Learning using unsupported services could not be supported.*
>
> *Partial solutions:*
> - *Provide an LD Service Vocabulary Registry. This would consist of a set of service definitions reduced to the variable parts of the Open Service Schema:*
>
> *<title>*
>
> *<role 1>*
>
> *<role 2>*
>
> *...*
>
> *<role n>*
>
> *<description> // probably needed to help potential users interpret the function and appropriate use of the service*
>
> *<source-url> // a location from which the service software can be downloaded*
>
> *<online-url> // one or more locations where an instance of the service (either commercial or free) is available for use by others*
>
> *<uddi-url> // the location of a WS service registry where the WSDL interface and locations of one or more instances of the service (either commercial or free) is available for use by others.*
>
> *It may be necessary to distinguish and support both a set-up WS interface as well as a runtime interface type: Web interface, WSRP, WS-interface, etc.*
>
> *The LD Service Vocabulary Registry would need to be at a well known location (e.g. OUNL or IMS). Everyone creating a service definition would be encouraged, first to search existing definitions and if one cannot be found, create a new one and post it to the Registry.*

- *Providing that services can either be downloaded and run locally, or located and used over the Internet, this problem should be reduced.*

But more is needed, since information will need to be exchanged between LD engines and services; the CCSI work seems very relevant in this respect. Work we tried to bring into QTIv2 on a kind of "harmonisation spec" is also relevant here:

- In terms of relating LD props to QTI variables, we might imagine something like

```
<imsldqtiharm:equivalent>
    <imsldqtiharm:property-ref idref="LD-Property-X"/>
    <imsldqtiharm:var-ref idref="QTI-Variable-Y"/>

</imsldqtiharm:equivalent>
```

- This avoids the need to move to the use of XML IDs for QTI variables (although that might be a good move anyway)

- We can also imagine that we might here specify which element of a multi-valued QTI variable is desired, and also whether data type coercion is required:

```
<imsldqtiharm:equivalent>
    <imsldqtiharm:property-ref idref="LD-Property-X"/>
    <imsldqtiharm:var-ref idref="QTI-Variable-Y" component="1"/>
        <imsldqtiharm:coerce-to "string"/>
</imsldqtiharm:equivalent>
```

- So if QTI-ITEM-1 is an assessment level collection of items, with an associated variable "SCORE" we can imagine the following questionnaire:

```
<imscp:manifest identifier="LD-QTI-CP">
<imscp:organizations>
 <imsld:learning-design identifier="Harmonization" uri="" level="B">
 ...
  <imsld:environments>
        <imsld:environment identifier="E-Test">
                <imsld:service identifier="Testservice">
          <imsldqtiharm:questionnaire>
                    <imsldqtiharm:equivalent>
                    <imsldqtiharm:property-ref
                              idref="LD-Intake-Result"/>
```

```
                        <imsldqtiharm:var-re
                                        idref="SCORE"/>
                    </imsldqtiharm:equivalent>
                      <imsldqtiharm:item idref="QTI-ITEM-1"/>
                </imsldqtiharm:questionnaire>
            </imsld:service>
              </imsld:environment>
    </imsld:environments>
    </imsld:components>
    <imsld:method>
</imscp:manifest>
```

So the spec we come up with will allow services which publish their presence to be found, configured, instantiated multiple times, linked to users in various roles (both roles define in a learning design and role in the conceptual world of the tool), started automatically, shut down automatically, given input at start-up or during operation, be requested to return output, either during operation or at shut down.

5.1.1. Proposed Solution

A solution is currently being researched, based on existing work done by IMS (the Tools Interoperability Guidelines), LAMS (Tool Contract API) and other organisations such as the Workflow Management Coalition. It is of note that project partners are also engaging with IMS in the development of IMS Tools Interoperability version 2.0, which may be a candidate specification for the connection protocol.

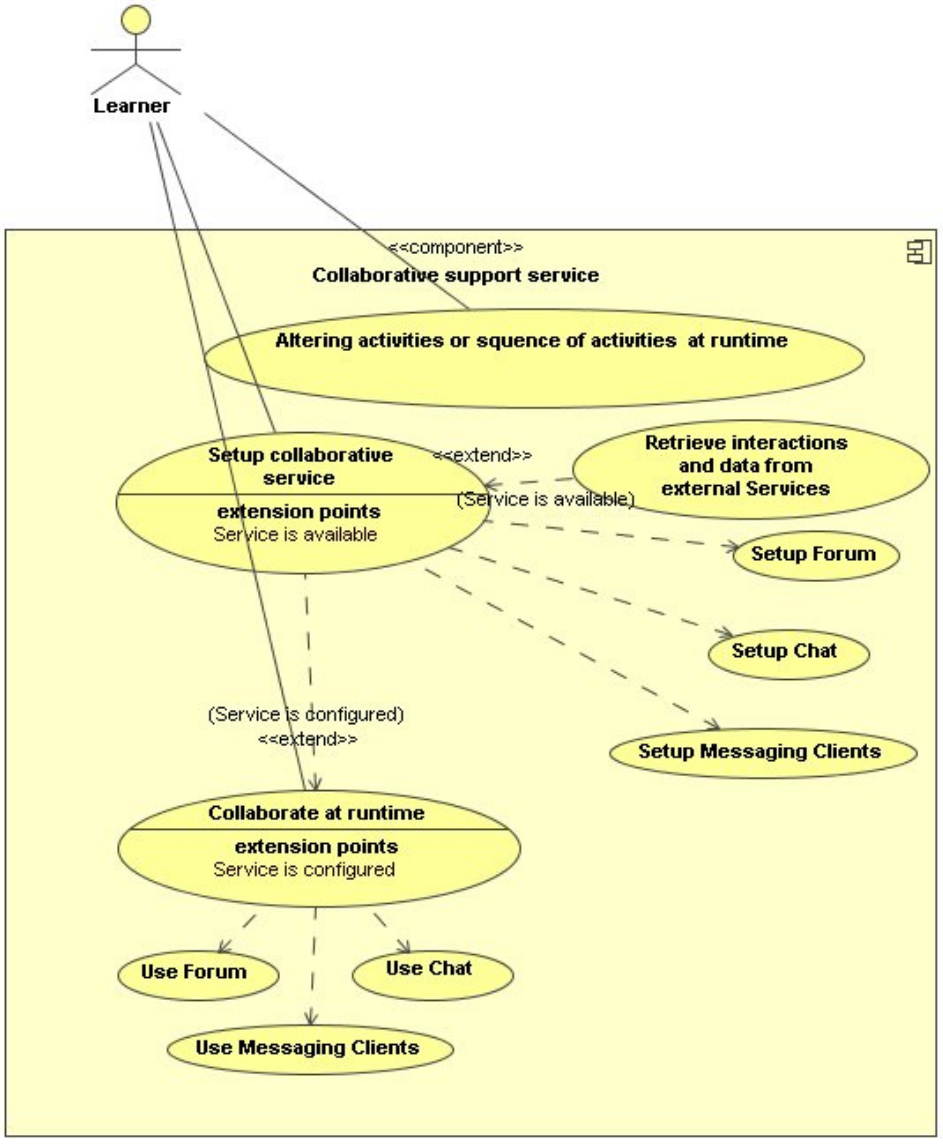
5.1.2. IMS LD Run-time Conclusions and Future Work

Future research work will concentrate on developing a generic connection protocol usable with a wider range of external tools. Currently a number of technologies have been identified as the basis for this work, including IMS Tools Interoperability, the LAMS tool contract API, the OASIS Service Provisioning Markup Language (SPML), and the Blackboard Building Blocks API. Any connection protocol needs to accomplish the key requirements of being able to launch and terminate a remote tool or service; additional requirements of monitoring, notification and intervention have also been identified.

Any proposed solution, however, also needs to address a range of related concerns, including provisioning and identity across organizational boundaries; these concerns require placing the connection protocol within a broader identity infrastructure. Identity is a key cross-cutting concern in many of the work areas of the project (e.g. repositories, client-server architecture, portfolios) and will need to be tackled in a consistent manner across the different work areas.

5.2. Use Cases for the Connection Protocol

Aside the TENCompetence IMS LD Authoring Tool and the TENCompetence Assessment Tool prototypes, the WP6 tool set include also the TENCompetence Run-time engine prototype. The latter implements the TENCompetence Connection Protocol. This will describe how IMS Learning Design Run-time Engines can be connected to external collaboration and communication services, in order to support pedagogical models that require the use of such services (e.g. collaborative learning). The following use cases aim at describing Learner's activities performed while using the Collaborative support service facilities and her/his interaction with the system.



Altering activities or sequence of activities at runtime	
Description	This use case describes how to alter a running e-learning process that is required to be changed
Exemplary Actors	Learner
Basic Flow	1. The User suspends the system run by invoking the 'Pause' facility 2. The User invokes the Editing Mode for performing the due changes. 3. The User selects what to alter (i.e. activities or sequence of activities). 4. The User performs and saves the due changes. 5. The User resumes the run from the interruption point.
Alternative Flow	• *Alternative Flow of Events:* On the Step 1 the System suspends automatically the run since a critic situation is detected. On the Step 2 the System offers the User the possibility to alter the faulting process by proposing to start the Editing Mode • *Alternative Flow of Events:* On the Step 5 the User resumes the run from starting point since it is not possible to do it from the interruption point. • *Alternative Flow of Events:* On the Step 5 after the changes are saved, the System resumes the run from the interruption point. • *Alternative Flow of Events:* On the Step 5 after the changes are saved, the System resumes the run from starting point since it is not possible to do it from the interruption point. • *Alternative Flow of Events:* On every Step the User could require Help information.
Pre-Conditions	1. The User has rights for altering activities or sequence of activities at runtime 2. A critic situation happens at run time and a specific UOL requires urgent changes (e.g. modifying / supplying a missing a

	definition; modifying / adding resources, activities, conditions)
Post-Conditions	1. The activities or sequence of activities at runtime are altered. 2. The e-learning running process has been resumed or re-started from scratch.
Specific Requirements	1. On the Step 1 the ability to suspend the process of activities and / or sequence of activities at runtime has to be provided in a simple way for the User, e.g. the 'Pause' facility should be invoked by a keystroke (i.e. 'Pause' to be associated to a specific key). 2. On the Step 2, the opening of the Editing Mode requires the launch of an editor / authoring tool as described in the WGA part. 3. On the Step 5 the ability to resume the process of activities and / or sequence of activities at runtime from the interruption point has to be provided in a simple way for the User, e.g. by the same procedure required for invoking the suspension (i.e. by stroking the same key associated to the 'Pause' facility).
Include	None

Retrieve interactions and data from external Services	
Description	This use case describes how interactions and data from external Services are retrieved from the Collaborative support service.
Exemplary Actors	Learner
Basic Flow	1. During the development of an activity the User accesses a page which has reference to a property. 2. The player (i.e. the System) gets the data from the URL defined in the property. 3. The data will be shown (i.e. made human-readable) in the user client in the way embedded on the page. 4. The User carries on the activity.
Alternative Flow	• *Alternative Flow of Events:* On the Step 1 the User initiates a service which refers to a property On the Step 3 the data will be handled by the service. • *Alternative Flow of Events:* On the Step 1 During the development of an activity a condition expression is met, in which a property is referred to, and evaluated. On the Step 3 the player (i.e. the System) will act differently according to the evaluation result.
Pre-Conditions	Data concerning the learners interaction with the service that has to be used in the further progression of the learning process are not available.
Post-Conditions	The needed data and interactions have been retrieved and used.
Specific Requirements	None
Include	None

Setup collaborative service	
Description	This use-case describes the process of setting up collaborative services at runtime
Exemplary Actors	Learner
Basic Flow	1. The User accesses the collaborative tools available in the System user interface. 2. The User selects and launches the appropriate collaborative tool 3. The System sets up the related collaborative service.
Alternative Flow	None
Pre-Conditions	1. During the development of an activity the User needs to cooperate/collaborate with other Learners in order to accomplish her/his tasks, therefore, • Use of collaboration tools (e.g. forum, chat, MSN Messenger, Yahoo Messenger, Google Gtalk, Skype chat, AOL IM, IRC, ICQ) are needed (i.e. the UOL design devises their use), and • Setup of a collaborative service is needed 2. Collaborative tools are available.
Post-Conditions	A collaborative service is set up.
Specific Requirements	A list of collaboration tools has to be available and accessible in the System user interface.
Include	None
Open Issues	Just in case the precondition on availability of the needed collaborative tool and / or service is skipped (e.g. in the first case the collaborative tool is not listed , while in the second case the tool is listed but cannot be accessed since the related service cannot be invoked and set up), the "Altering activities or sequence of activities at runtime" use case should be considered in the 'Error/Exception Flow of Events' section.

Setup Forum	
Description	This use-case describes the process of setting up a forum at runtime
Exemplary Actors	Learner
Basic Flow	1. The User accesses the collaborative tools available in the System user interface. 2. The User selects and launches the forum tool 3. The System sets up the collaborative service related to a forum.
Alternative Flow	None
Pre-Conditions	1. The use of a forum is needed (i.e. the UOL design devises its use), and 2. A forum is available.
Post-Conditions	A collaborative service for a forum is set up.
Specific Requirements	None
Include	None

Setup Chat	
Description	This use-case describes the process of setting up a chat at runtime
Exemplary Actors	Learner
Basic Flow	1. The User accesses the collaborative tools available in the System user interface. 2. The User selects and launches the chat tool 3. The System sets up the collaborative service related to a chat.
Alternative Flow	None
Pre-Conditions	1. The use of a chat is needed (i.e. the UOL design devises its use), and 2. A chat is available.
Post-Conditions	A collaborative service for a chat is set up.
Specific Requirements	None
Include	None

Setup Messaging Clients	
Description	This use-case describes the process of setting up a Messaging Client at runtime
Exemplary Actors	Learner
Basic Flow	1. The User accesses the collaborative tools available in the System user interface. 2. The User selects and launches the Messaging Client. 3. The System sets up the collaborative service related to a Messaging Client.
Alternative Flow	None
Pre-Conditions	1. The use of a Messaging Client is needed (i.e. the UOL design devises its use), and 2. A Messaging Client is available.
Post-Conditions	A collaborative service for a messaging client is set up.
Specific Requirements	None
Include	None

Collaborate at runtime	
Description	This use case describes the process of using the collaborative tools at runtime once their related services have been setup.
Exemplary Actors	Learner
Basic Flow	1. The collaboration tool launched in Step 2 of the "Setup collaborative service" use case opens. 2. The User uses the collaboration tool according to her/his needs.
Alternative Flow	None
Pre-Conditions	The collaborative services related to the collaboration tools the User wants to use have been set up (i.e. the tool is available and ready to be used).
Post-Conditions	The collaboration tool launched by the User is used.
Specific Requirements	None
Include	None

Use Forum	
Description	This use case describes the process of using a Forum at runtime once its related services has been setup.
Exemplary Actors	Learner
Basic Flow	1. The forum launched in Step 2 of the "Setup Forum" use case opens. 2. The User uses the forum according to her/his needs.
Alternative Flow	None
Pre-Conditions	The collaborative service related to the forum the User wants to use have been set up (i.e. the forum is available and ready to be used).
Post-Conditions	The forum launched by the User is used.
Specific Requirements	None
Include	None

Use Chat	
Description	This use case describes the process of using a Chat at runtime once its related services has been setup.
Exemplary Actors	Learner
Basic Flow	1. The Chat launched in Step 2 of the "Setup Chat" use case opens. 2. The User uses the Chat according to her/his needs.
Alternative Flow	None
Pre-Conditions	The collaborative service related to the Chat the User wants to use have been set up (i.e. the Chat is available and ready to be used).
Post-Conditions	The Chat launched by the User is used.
Specific Requirements	None
Include	None

Use Messaging Clients	
Description	This use case describes the process of using a Messaging Client at runtime once its related services has been setup.
Exemplary Actors	Learner
Basic Flow	1. The Messaging Client launched in Step 2 of the "Setup Messaging Clients" use case opens. 2. The User uses the Messaging Client according to her/his needs.
Alternative Flow	None
Pre-Conditions	The collaborative service related to the Messaging Client the User wants to use have been set up (i.e. the Messaging Client is available and ready to be used).
Post-Conditions	The Messaging Client launched by the User is used.
Specific Requirements	None
Include	None

5.3. Requirements

5.3.1. Technical Requirements

This section sets out the key technical requirements for the connection protocol. The following terms are used throughout:

Collaboration Service (CS)

The system responsible for the provision of a collaboration interface hosted separately from the system responsible for the management of overall user activity. Examples of collaboration services include a whiteboard, shared document authoring, and instant messaging applications.

Activity Management Service (AMS)

The system responsible for the coordination of user activity and collaboration services. An example of an Activity Management Service is the CopperCore Service Integration component.

Authoring Application

The system that enables the authoring of processes that can be enacted by the Activity Management Service.

Service Registry

The system responsible for maintaining a directory of Collaboration Services usable by both the Activity Management Service and Authoring Application.

5.3.2. Functional Requirements

Registration

The Collaboration Service must be able to register itself with a Service Registry so that it can be discovered and incorporated into a process managed by the Activity Management Service and/or the Authoring Application.

To this end, each Collaboration Service needs to provide:

- Basic descriptive metadata, such as a title and description
- Version information
- A means of identifying service endpoints and protocols needed for service invocation.

For the last of these requirements, two approaches are possible: either the service exposes all endpoints and protocols using something like the Web Service Description Language (WSDL), or provides a URL to enable dynamic auto-discovery. The opposite operation, de-registration, should also be supported by the Service Registry.

Authoring support

The Collaboration Service must expose certain capabilities to support the authoring process. Specifically, the Collaboration Service must expose the parameters that can be set to configure the service. These can be simple descriptive elements (such as guidance text to show the user, framing questions etc.), presentation options (such as styles), or parameters that control the service logic (such as various processing options).

The Collaboration Service also needs to expose permission sets so that these can be mapped using the Authoring Application to roles defined within the learning design., and users allocated appropriate permissions when the Collaboration Service is launched.

Launch

The Collaboration Service must be capable of instantiating services for users as instructed by the Activity Management System. The Collaboration Service must provide an interface that supports the following:

- Provisioning of services with users, permissions, and activity context
- Configuration of the service

Configuration information is also described under Authoring Support. Provisioning information needs to indicate:

- The group context within which the service is provisioned (i.e. the common context of the users)
- The individual user information and permissions to be granted by the service instance to each of those users

For example, an AMS may launch a Chat CS with a group chat room context, within which most users are granted the Chat permission through membership of the Learner role, and one user is granted the Moderate permission through membership of the Teacher role. The AMS needs to communicate this context to the CS and provision it accordingly. Upon launch, the CS needs to inform the AMS how to access the individual views of the CS instance (e.g. the shared chat room, whiteboard that provides context for user sessions) for each user, and to be able to correlate subsequent service calls to the specific CS instance.

Monitoring

The CS needs to be able to expose the state of an instance, either in terms of individual activity (sessions) or group activity (context) to enable the tracking and monitoring of the service by the process owner. At its simplest this could be managed entirely by the CS, with provision to the AMS of a URL to reach the monitoring user interface.

Intervention

The CS needs to support the dynamic intervention by the process owner into a running service instance; for example, for a teacher to alter the difficulty level of an assessment

either for an individual, or for all sessions in a context. The CS should expose an interface for an instance that enables the AMS to set configuration properties dynamically during runtime in addition to at launch.

Event Propagation

It may be necessary for the CS to notify the AMS of events affecting the overall flow of activity external to the CS instance; in LD terms it may need to set a global property. To support this, the AMS needs to provide an interface that can be accessed by the CS at runtime.

Termination

The CS needs to provide a mechanism whereby the AMS can terminate an instance of a service, either for an individual session or for a complete context and all its contained sessions. On termination, the CS must provide outcome information to the AMS, such as completion state and other variables relevant to the overall Learning Design. Depending on its configuration, a CS may continue to allow access to a CS instance after termination, however this access may be determined to be read-only (a "locked" service). This may be determined using either the Launch or Intervention interfaces.

Completion

The AMS needs to provide a mechanism whereby the CS can indicate completion of activity within a CS instance. The AMS must provide an interface where the CS can tell the AMS the session that completed, any outcome information, and for the AMS to trigger some kind of placeholder view to be presented to that user. As with Termination, completion may occur either at the level of an individual session, or all sessions within a context.

Depending on its configuration, a CS may continue to allow access to a CS instance after completion, however this access may be determined to be read-only (a "locked" service). This may be determined using either the Launch or Intervention interfaces.

Export

At some point after completion of termination, the AMS may require the complete state of the CS instance to be made available as a record of activity either for user portfolios or for archiving. This may be in a static form (e.g. a PNG of a whiteboard, a text transcript of a chat). The CS should provide an export interface to support this.

Non-functional Requirements

Browser client environment

The solution must be capable of operating where the AMS supports a browser-based user interface (e.g. CCSI and SleD).

Rich client environment

The solution must be capable of operating where the AMS supports a rich client (e.g. Eclipse desktop client)

Session management

The CS and AMS must be capable of managing and correlating persistent sessions.

Single sign-on

Users should not need to sign-on independently to the CS.

Versioning

Collaboration Services need to support version control, and expose version information to the Service Registry and AMS.

Standards-based

Wherever possible and practical, the protocols used by CS and AMS should conform to existing standards and specifications.

Privacy

Where services are provisioned across organisational boundaries, privacy concerns of users are respected.

Multi-organisation

The AMS and CS should not need to be located within the same organization (learning network provider). A user should be able to create and share an activity using a CS hosted by an external provider.

6. Connection protocol: requirements and services

6.1. Introduction

This document sets out the key technical requirements for the connection protocol. The following terms are used throughout:

Collaboration Service (CS)

The system responsible for the provision of a collaboration interface hosted separately from the system responsible for the management of overall user activity. Examples of collaboration services include a whiteboard, shared content creation, and instant messaging applications. Collaboration Services have a wide range of possible implementations, from complete standalone servers to simple client-side applications ("widgets").

Activity Management Service (AMS)

The system responsible for the coordination of user activity and collaboration services. An example of an Activity Management Service is the CopperCore Service Integration component.

Authoring Application

The system that enables the authoring of processes that can be enacted by the Activity Management Service.

6.2. Functional Requirements

R1: Registration

The Collaboration Service must be able to register itself so that it can be discovered and incorporated into a process managed by the Activity Management Service and/or the Authoring Application.

To this end, each Collaboration Service needs to provide:
1. Basic descriptive metadata, such as a title and description
2. Version information
3. A classification of the kind of services, e.g. messaging, whiteboard etc.

The opposite operation, de-registration, should also be supported.

R2: Authoring support

The Collaboration Service must expose certain capabilities to support the authoring process.

Specifically, the Collaboration Service must expose the parameters that can be set to configure the service. These can be simple descriptive elements (such as guidance text to show the user, framing questions etc.), presentation options (such as styles), or parameters that control the service logic (such as various processing options).

The Collaboration Service may also need to expose permission sets so that these can be mapped using the Authoring Application to roles defined within the learning design. Users can then be allocated appropriate permissions when the Collaboration Service is launched.

R3: Launch

The Collaboration Service must be capable of instantiating services for users as instructed by the Activity Management System.

R4: Runtime Service Configuration

The Collaboration Service must provide an interface that supports its configuration, using one of the following models:

1. **Early binding**: the Activity Management System sends configuration details with the launch request
2. **Late binding**: the Collaboration Service launches in a default configuration and requests configuration information from its context as required

Configuration information is also described under Authoring Support.

R5: Personalisation

The Collaboration Service must be capable of being personalised for a particular user. Specifically, the service must be capable of requesting configuration information (see above) within the context of a particular instance (user session). This can be accomplished using one of several models:

1. **Provisioning**: the Activity Management Service sends user information to the Collaboration Service with the launch request
2. **Shared session**: the Collaboration Service is instantiated within the same session context as the activity it supports, and shares contextual information
3. **Contextualised session and late binding**: The Collaboration Service is instantiated within a contextualised session; i.e., launched with a contextual identifier which of itself conveys no user information, and requests user information as required from appropriate sources.

The last option would appear to have the fewest complications in terms of security and privacy concerns.

R6: Shared State

The Collaboration Service must be capable of sharing state across several users that operate in the same context, such as a group within a learning activity, for example the shared state for an instant messaging service would be the history of conversation among the participants. The Collaboration Service needs to be able to access a persistent shared state accessible only by participants.

R7: Monitoring

The CS needs to be able to expose the state of an instance or collection of instances, either in terms of individual activity (sessions) or group activity (context) to enable the tracking and monitoring of the service by the process owner.

At its simplest this could be managed entirely by the CS, with provision to the AMS of a URL to reach the monitoring user interface.

R8: Intervention

The CS needs to support the dynamic intervention by the process owner into a running service instance; for example, for a teacher to alter the difficulty level of an assessment either for an individual, or for all sessions in a context. The CS should expose an interface for an instance that enables the AMS to set configuration properties dynamically during runtime in addition to at launch. This may operate directly upon the shared state and individual instance states of the CS.

R9: Event Propagation

It may be necessary for the CS to notify the AMS of events affecting the overall flow of activity external to the CS instance; in LD terms it may need to set a global property. To support this, the AMS needs to provide an interface that can be accessed by the CS at runtime.

R10: Termination

The CS needs to provide a mechanism whereby the AMS can terminate an instance of a service, either for an individual session or for a complete context and all its contained sessions. On termination, the CS must provide outcome information to the AMS, such as completion state and other variables relevant to the overall Learning Design.

Depending on its configuration, a CS may continue to allow access to a CS instance after termination, however this access may be determined to be read-only (a "locked" service).

R11: Long-running Sessions

A CS may need to maintain a long-running activity, keeping session and context states and resuming instances outside of a single browser session.

R12: Completion

The AMS needs to provide a mechanism whereby the CS can indicate completion of activity within a CS instance. The AMS must provide an interface where the CS can tell the AMS the session that completed, any outcome information, and for the AMS to trigger some kind of placeholder view to be presented to that user. As with Termination, Completion may occur either at the level of an individual session, or all sessions within a context. Depending on its configuration, a CS may continue to allow access to a CS instance after completion, however this access may be determined to be read-only (a "locked" service).

R13: Export

At some point after completion of termination, the AMS may require the complete state of the CS instance to be made available as a record of activity either for user portfolios or for archiving. This may be in a static form (e.g. a PNG of a whiteboard, a text transcript of a chat). The CS should provide an export interface to support this.

6.3. Non-functional Requirements

R14: Browser client environment

The solution must be capable of operating where the AMS supports a browser-based user interface (e.g. CCSI and SleD).

R15: Rich client environment

The solution must be capable of operating where the AMS supports a rich client (e.g. Eclipse desktop client)

R16: Session management

The CS and AMS must be capable of managing and correlating persistent sessions.

R17: Single sign-on

Users should not need to sign-on independently to the CS.

R18: Versioning

Collaboration Services need to support version control, and expose version information to the AMS.

R19: Standards-based

Wherever possible and practical, the protocols used by CS and AMS should conform to existing standards and specifications.

R20: Privacy

Where services are provisioned across organisational boundaries, privacy concerns of users are respected.

R21: Open Source

Solutions should use open-source components.

R22: Accessibility

Solutions should not preclude use by people with accessibility requirements.

6.4. Non-Requirements

The following requirements have been determined to be out of scope.

R23: Multi-organisation

The AMS and CS should not need to be located within the same organization (learning network provider). A user should be able to create and share an activity using a CS hosted by an external provider.

6.5. Proposed Solution

The following were considered as providing a potential basis for the solution architecture:

1. The IMS Tools Interoperability Guidelines, v1.0
2. The IMS Learning Tools Interoperability Specification, v 2.0 (under development)
3. The LAMS Tool Contract
4. The W3C Widget Protocol, v1.0 (under development)

After initial exploration and experiments it was decided to work on an architecture following the general approach of the W3C Widget Protocol. In particular, it was felt that this would enable a much simpler and more lightweight set of requirements for collaboration services to meet, and enable existing widgets developed for the Apple, Microsoft, and Yahoo platforms to be fairly easily ported to operate within an LD environment.

While the LAMS tool contract is a more mature technology, it was felt that many of the underlying assumptions of the API are dependent on the LAMS system, particularly the tight coupling of user information within a single application server, and would be more difficult to make generic, component based solution.

The IMS Tools Interoperability Guidelines specify an architecture that is much more like a conventional LMS setup, and offered no capabilities for monitoring, or shared state. It also requires considerable effort on the part of tool authors to implement a range of SOAP Web Services.
The following is an overview of the proposed architecture.

6.6. Widget Connection Protocol

Architecture

The overall architecture is shown in Figure 14.

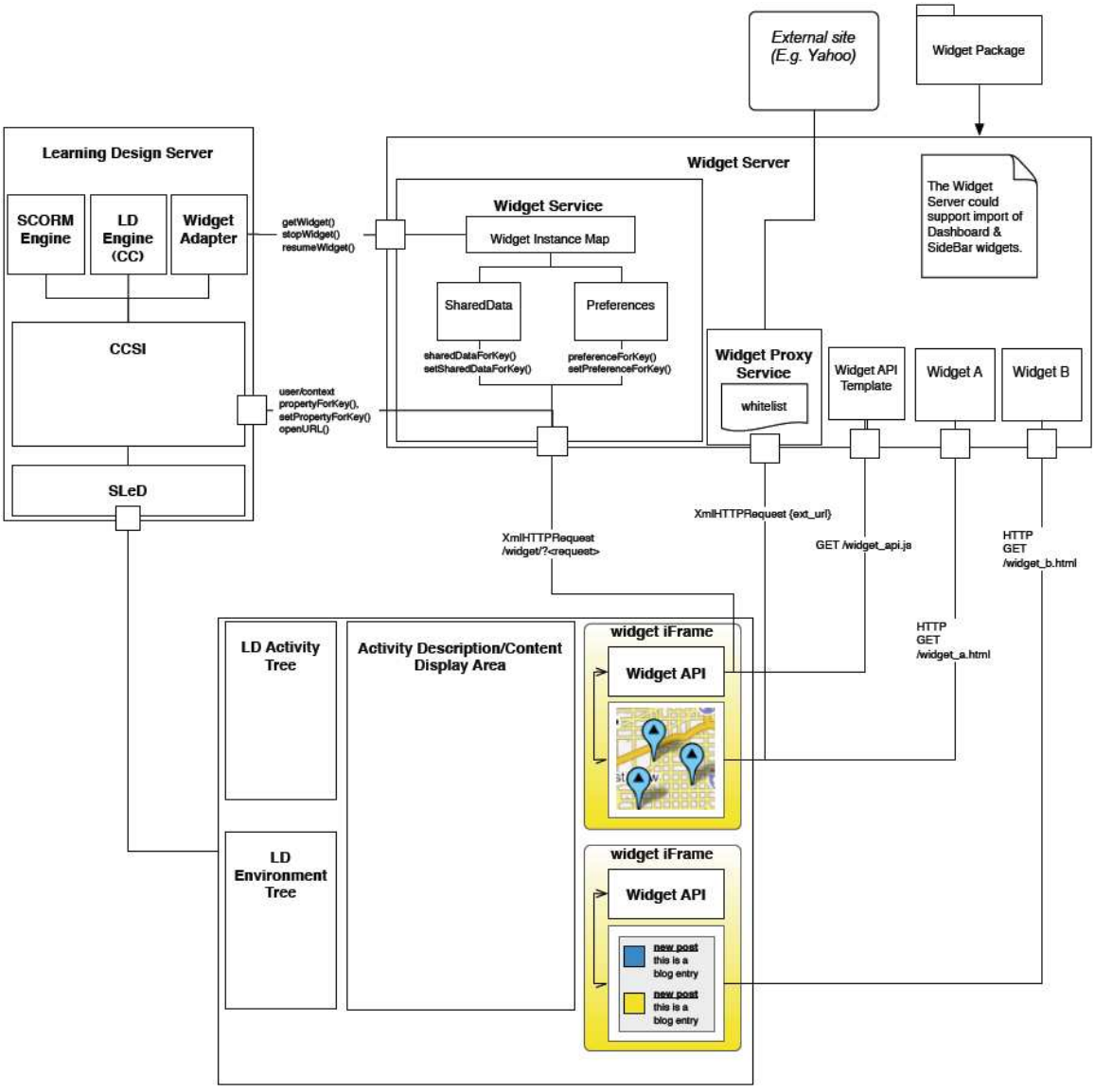


Figure 14. ***Overview of solution architecture***

A **Widget** is a self-contained application for displaying and updating remote data that can be deployed as part of a broader platform. *Widgets* may operate singly or may act collaboratively using shared states. *Widgets* are deployed on, and instantiated by, a *Widget Server*. Each *Widget Instance* can communicate with the *Widget Server* through the *Widget API*. (The term "widget" is introduced in this document to replace the ambiguous term "service" used in Learning Design).

The **Widget Server** is the platform which provides a *Widget Service*, and deploys one or more *Widget Instances* and their corresponding *Widget API*, typically through serving an iFrame containing a page with JavaScript references to the *Widget* JavaScript and the *Widget API* JavaScript. The Widget Server can offer a range of *Widgets* that have been imported and registered using standard Widget Packages (such as the W3C, Apple, Windows and Yahoo Widget Package formats).

The **Widget Service** is the service provided by the *Widget Server* that responds to calls made to the *Widget API* and instantiates and manages widgets in response to requests from external applications, such as CCSI.

The **Widget API** is the local JavaScript API accessible to each *Widget Instance* that enables it to communicate with the *Widget Server*.

A **Widget Instance** is a *Widget* instantiated within a particular user's browser.

A **Widget Proxy Service** is a service that supports Widgets to make calls to remote services, avoiding the Same Origin Policy of the user's browser.

6.7. The Widget Service

The Widget Service offers two main capabilities:

- it fulfils requests from the Widget API
- it supports requests for new Widget Instances

The former capability essentially maps onto the Widget API itself (see below).

To prevent inappropriate access to personal information, the Widget Service should offer its services using TLS-encrypted channels (HTTPS).

Widget Instantiation

To instantiate a widget, the Learning Design Server must obtain a reference to a WidgetAdapter instance using CCSI. The WidgetAdapter provides the following methods:

getWidget(String user_id, String run_id, String service_id, String type): [String, String, String]

The getWidget() method instantiates (if necessary) and returns the URL and the height and width in pixels for a *widget* of the given *type*. For example:

```
IWidgetAdapter adapter = Dispatcher.getAdapter('WidgetAdapter');
String[3] widget = adapter.getWidget(userid,runid,servid,'chat');
url = widget[0];
height = widget[1];
width = widget[2];
```

The getWidget method results in the following sequence of actions by the Widget Service:

1. The Widget Service checks the WidgetInstances collection to see if there is an existing Widget Instance matching the supplied parameters. If one exists, skip to step 6.
2. Search the Widgets collection to locate a matching *Widget* for the type specified. If none exists, throw a WidgetTypeNotSupportedException
3. Create a new Widget Instance using the supplied parameters, and the Widget ID of the selected *Widget*
4. Add a randomly-generated Nonce value to the Widget Instance, and generate an id_key from the Widget Instance using the Secure Hashing Algorithm (SHA_1).
5. Store the Widget Instance and its id_key in the WidgetInstances collection
6. Generate a URL for the Widget Instance, using the URL value of the Widget from the Widgets collection, plus the id_key from the WidgetInstances collection, the Widget Service URL as *url*, and Widget Proxy Service URL as *proxy*. E.g.:

http://my.widget.server/widgets/converter/index.html?id_key=134ab324ed3423b23ff1232d&url=http://my.widget.server/widgets/service&proxy=http://my.widget.server/proxy

7. Obtain the height and width of the *Widget* from the Widgets collection.
8. Return the URL, height, and width as a String Array.

The Learning Design Server should use the supplied URL within an IFrame in the user's application, set to the height and width specified.

stopWidget(String user_id, String run_id, String env_id, String type)

The stopWidget() method is equivalent to the Lock() method in the *Widget API*. The Widget is no longer capable of having its shared data altered.

resumeWidget(String user_id, String run_id, String env_id, String type)

The opposite of stopWidget().

6.8. The Widget API

Widget Object

The Widget Object is used by a *Widget Instance* to access the functions of the *Widget API*. It is instantiated automatically within the Widget Instance environment, and can be accessed simply by:

widget.{property||method()}

The Widget Object is based on the W3C draft specification for Widgets [1,2], and the Apple Dashboard API [3].

Properties

id_key

The unique identifier for this *Widget*. This read-only property contains a string value that is unique among all of the instances of a single *Widget*. This value is assigned by the *Widget Server*.

proxy

The URL of the *Widget Proxy Service* to use for XmlHttpRequests made by this *Widget Instance*. The *Widget Instance* should use this URL to construct requests, in the form:

{proxy}?target={url}

For example, using Prototype.js:

```
var url = widget.proxy + '?url=' +
encodeURIComponent('http://www.google.com/searchq=Prototype')+'&id_key='
+widget.id_key;

new Ajax.Request(url, {

    method: 'get',

    onSuccess: refreshWidget(transport)

    });

}
```

Using the proxy avoids cross-domain scripting restrictions in place in the users browser (the Same Origin Policy). Note that the *Widget Server* may use whitelists and/or blacklists to determine whether to carry out the request; the proxy must return a 401 HTTP status code if it decides to deny the request.

Note that the *Widget Proxy Service* must not accept requests that do not contain the id_key of the *Widget Instance*.

onShow

Contains the callback function to be called when the widget instance is displayed. This callback shall be triggered whenever the user hides and then shows a particular widget in the interface. Widgets may want to use this function to trigger a refresh or to resume suspended processing. For general initialization, the Widget should not rely on the onShow function as it may not be triggered on initial rendering.

The Widget API implementing this function may use any appropriate method for supporting this function, including periodic polling of the Widget Service using a timer and XmlHttpRequest functions.

A Widget Instance that wishes to be notified when it is shown should implement a function for the callback.

For example:

```
widget.onShow = handleShow;
function handleShow(){
    init();
}
```

onHide

Contains the callback function to be called when the widget instance is hidden, either when minimized in the view or when the user moves to a different activity (but may track back and have the widget show once more). Widgets may want to use this function to suspend processing.

The Widget API implementing this function may use any appropriate method for supporting this function, including periodic polling of the Widget Service using a timer and XmlHttpRequest functions.

A Widget Instance that wishes to be notified when it is hidden should implement a function for the callback. For example:

```
widget.onHide = handleHide;

function handleHide(){
    refreshHandler.stop();
    setAwayMessage("away");
}
```

onLocked

Contains the callback function to be called when the widget instance is locked using the widget.lock() method or the Widget Service stopWidget() method.

The Widget API implementing this function may use any appropriate method for supporting this function, including periodic polling of the Widget Service using a timer and XmlHttpRequest functions.

A Widget Instance that wishes to be notified when it is locked should implement a function for the callback. For example:

```
widget.onLocked = handleLocked;
function handleLocked(){
```

```
    refreshHandler.stop();
    grayOutEverything();
}
```

onUnlocked

Contains the callback function to be called when the widget instance is unlocked using the widget.unlock() method or the Widget Service resumeWidget() method.

The Widget API implementing this function may use any appropriate method for supporting this function, including periodic polling of the Widget Service using a timer and XmlHttpRequest functions.

A Widget Instance that wishes to be notified when it is unlocked should implement a function for the callback. For example:

```
widget.onUnlocked = handleUnlocked;


function handleUnlocked(){
    refreshHandler.start();
    enableInput();
}
```

onSharedDataUpdate

Contains the callback function to be called when any of the shared data associated with the widget instance is modified. The Widget API implementing this function may use any appropriate method for supporting this function, including periodic polling of the Widget Service using a timer and XmlHttpRequest functions.

A Widget Instance that wishes to be notified of shared data changes should implement a function for the callback. For example:

```
widget.onSharedDataUpdate = handleUpdate;


function handleUpdate(keys){
    for (String key:keys){
         if(key.equals("chat")                    sharedDataForKey("chat",
refreshChatDisplay());
     }
}
```

The callback method will be called with an array of keys for the shared data that has been updated as its parameter.

Methods

Asynchronous methods

All methods that return data are **asynchronous**. This means that rather than expecting a return value from the API call, you should instead pass a callback object (typically a function reference) so that the API can invoke your script with the return value when the asynchronous HTTP request returns a completed response. This also applies in the case of 'set' methods; in this case, callback objects can be used to trigger actions based on the success or failure of a call to set a value.

Preferences

'Preferences' is used for persisting local data for the *Widget Instance*, and are not shared among *Widget Instances*. This should be used for storing and retrieving local configuration information such as user-entred data.

Shared Data

'Shared Data' is used for persisting shared data usable by all *Widget Instances* that have been instantiated within a single *context*; i.e. a specific 'environment' within a Learning Design 'run'. This should be used for sharing state among collaborating widgets, such as the content of a chat session, or the state of a game.

UserProperty

User is used for accessing information about the current user within the current context (i.e. their Learning Design role-part instance). Depending on the setup, some UserProperty values may also be set by the *Widget Instance*.

ContextProperty

Context is used for accessing information about the current context (Learning Design 'run'). Depending on the setup (e.g. IMS Level A vs B/C), some ContextProperty values may also be set by the *Widget Instance*.

Keys

Keys identify value spaces within a dictionary.

preferenceForKey(String key, Callback object)

The preferenceForKey() method takes a String argument, key. When called, this method shall query the service for the object reference by the key, and then invoke the Callback function provided with an object that has previously been stored with the setPreferenceForKey method, or *null* if the key does not exist. Example:

```
widget.preferenceForKey('name',showName());


function showName(name){
   update('name_label',name);
}
```

setPreferenceForKey(String key, Object preference)

The setPreferenceForKey() method takes two arguments, preference and key. When called, this method shall store the object in the preference argument with the key named in the key argument for later retrieval using the preferenceForKey() method. If the setPreferenceForKey() method is called with the value null in the preference argument, the key identified in the key argument shall be deleted.

sharedDataForKey(String key, Object callback)

The sharedDataForKey() method takes a String argument, key. When called, this method shall return an object that has previously been stored with the setSharedDataForKey method, or *null* if the key does not exist.

setSharedDataForKey(string key, Object data)

The setSharedDataForKey() method takes two arguments, data and key. When called, this method shall store the object in the data argument with the key named in the key argument for later retrieval using the sharedDataForKey() method. If the setSharedDataForKey() method is called with the value null in the preference argument, the key identified in the key argument shall be deleted. This method will trigger the onSharedDataUpdate property to be set True for all Widget Instances that share the same data storage context.

appendSharedDataForKey(string key, Object data)

The appendSharedDataForKey() method takes two arguments, data and key. When called, this method shall append the object in the data argument to object with the key named in the key argument for later retrieval using the sharedDataForKey() method. This

method will trigger the onSharedDataUpdate property to be set True for all Widget Instances that share the same data storage context.

lock()

The lock() method locks the shared data for the *context* of the *Widget Instance* and prevents this and any other Widget Instances writing to it via the setSharedDataForKey() or appendSharedDataForKey() methods.

unlock()

The opposite of lock(), the unlock() method lets *Widget Instances* write to shared data for the *context* of this *Widget Instance* via the setSharedDataForKey() and appendSharedDataForKey() methods.

userPropertyForKey(String key, Callback object)

Operates in the same manner as preferenceForKey(), but the request should instead be routed to the AMS by the Widget Service for resolution against the Learning Design Dossier.

setUserPropertyForKey(String key, Object property)

Operates in the same manner as setPreferenceForKey(), but the request should instead be routed to the AMS by the Widget Service for resolution against the Learning Design Dossier.

contextPropertyForKey(String key, Callback object)

Operates in the same manner as preferenceForKey(), but the request should instead be routed to the AMS by the Widget Service for resolution against the Learning Design Run Properties.

setContextPropertyForKey(String key, Object property)

Operates in the same manner as setPreferenceForKey(), but the request should instead be routed to the AMS by the Widget Service for resolution against the Learning Design Run Properties.

openURL(String url)

The openURL() method on the widget object takes a String as an argument. When this method is called with a valid URL as defined by [RFC3987], the URL should be opened by the Widget Server on which the widget runs.

For example, the Widget Server may open the url in a new browser window, or in a frame within the existing window

6.9. Widget Support Service

The Widget Support Service is offered by the AMS (in this case CCSI) to support calls made from Widgets to the Widget API that have a concern that is not handled by the Widget Server, such as requests for user properties and context properties. The Widget Support Service should offer the following methods:

userPropertyForKey(String user_id, String run_id, String key): Object

setUserPropertyForKey(String user_id, String run_id, String key, Object property): boolean

contextPropertyForKey(String context_id, String run_id, String key): Object

setContextPropertyForKey(String context_id, String run_id, String key, Object property): boolean

openURL(String url)

These methods are the server-side implementations of the API methods of the same name. Note that the Widget Server needs to lookup the relevant identifier fields from the hashcode of the invoking Widget Instance to generate the calls to the Widget Support Service in order to correlate the activity context with the Widget context.

6.10. Requirements Match

The proposed solution matches the requirements as described below.

Functional requirements

R1: Registration. The Widget Server provides a 'registry' function to enable widgets to be registered and be made available.

R2: Authoring support. As Widgets are instantiated on the basis of availability in a generic way, and configuration is handled via late binding, it is not necessary to incorporate configuration into the authoring process. This may be added later as functionality is identified.

R3: Launch. The Widget Server offers launch capabilities to the AMS.

R4: Runtime Service Configuration. The Late Binding model is supported.

R5: Personalisation. The Contextualised session and late binding model is supported.

R6: Shared State. The Shared Data API and shared data implementation of the Widget Server offers this functionality.

R7: Monitoring. This is not currently supported, but could be envisaged as an add-on to the Widget Server that allowed privileged inspection of instance and shared data properties.

R8: Intervention. The ability to suspend or lock an activity is offered by the Widget Server. Specific intervention in the form of direct manipulation of shared data or instance data is currently delegated to individual Widgets. E.g., a generic messaging widget may have a "moderator facade" that activates if the widget identifies the user as having the "moderator" role and offers additional functionality.

R9: Event Propagation. This is offered by the AMS to the Widget Server.

R10: Termination. This is offered by both the Widget Server to the AMS, and also via the Widget API, as a "lock()" method.

R11: Long-running Sessions. The Widget Server is designed to be able to continue long-running sessions by persisting instances identified by unique hashcodes.

R12: Completion. No mechanism is currently given to notify the AMS of completion. This may be added to the Widget Support Service.

R13: Export. The Widget Server may offer this capability, although it is currently undefined.

Non-functional Requirements

R14: Browser client environment. This is the default architecture.

R15: Rich client environment. This is not precluded, provided that the rich client offers HTML and JavaScript capabilities.

R16: Session management. The hashcode method provides this capability.

R17: Single sign-on. As the Widget is instantiated within an iFrame in an existing application window, no sign-on is required. The hashcode used for instantiation doubles up as a token authenticating use.

R18: Versioning. Version information should be included in the widget manifest and included in the registry maintained by the Widget Server.

R19: Standards-based. The solution uses Internet standards supplemented by APIs derived from the developing W3C Widget Specification([1], [2]).

R20: Privacy. The late binding model ensures there is no delegation of user information concerns, enhancing privacy. Also, no user information is automatically transacted at instantiation. Where information is transferred, e.g. as a result of a request from the Widget to the AMS brokered by the Widget Service, this information is transmitted encrypted.

R21: Open Source. The solution will be constructed as a set of open-source components.

R22: Accessibility. The solution uses technologies such as JavaScript and AJAX, which, while not inherently non-accessible, pose some challenges to developers to implement in an accessible way.

6.11. Development Strategy and conclusions

The widget service is what a Learning Design runtime system (CCSI) will want to initially connect to. The development of this involves creating the infrastructure contained within the w*idget service* box in the **Fout! Verwijzingsbron niet gevonden.**. At first this would include storing dummy widgets, creation of the widget instance mechanism and serving the correct data format to a call from a **dummy LD runtime system**. This is followed by implementing the shared data & preferences part. Finally the widget once loaded into the web player, may wish to be able to obtain properties from the Learning Design runtime system. To do this would involve a call-back system from the widget server which would **query CCSI.**

The next part is to incorporate and expand on the simple “dummy “calls, so that they can be made from CCSI. This means creating a new service or **widget adapter** for CCSI. It enables the connection between the Learning Design Engine and the widget server – for real. At this stage the web player would have to be **updated (SLeD)** so that it can handle the new types of connection/content.

Once the content can be loaded into the browser, it needs to be able to communicate with the widget server (from within SLeD). This means the next task is to develop the full widget javascript API. This will be a **downloadable javascript library**, available from the widget server. The calls made from the widget API then need a handler – to process the calls made from the client (SLeD) and **deal with the requests made onto the widget service.**

It will then be necessary to implement the **widget proxy service**. This is needed because a widget may wish to talk to a third server to obtain data. Due to security restrictions found in all browsers, this is not allowed by default. To circumvent this, we will need to

develop a handler mechanism which translates calls so that they appear to come from the same domain.

We also need to allow the system to **import existing** Mac dashboard widgets and windows Vista sidebar gadgets. We need a system to handle this not only to parse and import these, but also once running, because they both rely on specific javascript extensions – found only within the specific operating systems it will need to handle them while running too. These extensions need to be added somehow to the widget adapter, so that they run as if they were running in their native environment – i.e. in the dashboard, or sidebar.

We believe that this is a very promising approach to providing a generalisable solution for the provision of runtime services for IMS LD. Consequently it will constitute the principal line of work in implementing interoperable runtime services in the coming period of project work.

7. Summary and conclusions

As described in the main text of this deliverable, the Run-time Component task sets out to build on existing Open Source runtime engines to ensure that effective and usable systems are available to run the Units of Learning and Units of Assessment which are developed by the project. Two principal challenges were identified:

c) There is no runtime environment which is compliant with the latest version of the IMS QTI specification, version 2.1.

d) There is a lack of flexible runtime services (e.g. a forum, or a chat, run external learning objects and tests, etc.) which can be launched from an LD player (typically a system based on the CopperCore LD Engine).

Both these issues have been addressed in this first phase of project work.

Firstly, as described in section 4, the APIS runtime environment for IMS QTI which was developed with funding from the UK eFramework has been extended to support IMS QTI 2.1. This was a major undertaking as the structure of the specification has been radically changed in V. 2.1. In the work reported above we have described how this has been achieved. This functionality is now ready for integration, and is available for download at http://sourceforge.net/projects/newapis.

The second challenge is more fundamental, because it has resisted satisfactory solution since the approval of the IMS LD specification. To address this a first version of the runtime system was developed tested and released, as described in section 3. This builds on the most advanced currently available system for delivering services with IMS LD, which involves integrating services in the CCSI layer of CopperCore. The first service to be integrated was a SCORM player, making it possible for the two leading interoperability specifications for learning flows (IMS LD and SCORM) to work together. This work has been completed, tested and published as Open Source at http://tencompetence.cvs.sourceforge.net/tencompetence/wp6/

This experience, however, showed that the integration of additional services would be a time consuming process. Given the large number of services which it would be desirable to use with IMS LD, it was therefore concluded that it was not a generalisable approach to service integration, as required in TENCompetence. Consequently an entirely new approach to a LD service integration has been established, making use of Widgets as a shared service. A specification for this connection protocol has been developed, and is described in section 5 & 6 above. Implementation work is currently underway on a trial server which will be delivered in D6.2.

Work in the next phase of project work will concentrate on implementation of a proof of concept server to make widget services available within the SLeD and CopperCore environment, with a first version scheduled for month 24.

References

[1] W3C Widget Specification Working Draft v1.0, http://www.w3.org/TR/widgets/

[2] W3C Widget Requirements v1.0, http://www.w3.org/TR/widgets-reqs/

[3] Apple Dashboard Developer Reference, http://developer.apple.com/documentation/AppleApplications/Reference/Dashboard_Ref/index.html

[4] Windows Sidebar Reference, Microsoft Corporation, 2006. Available at http://msdn2.microsoft.com/en-us/library/aa965853.aspx

[5] Yahoo! Widget Engine Reference 3.1 Reference Manual Yahoo! Inc., April 14, 2006. Available at http://widgets.yahoo.com/gallery/dl_item.php?item=WidgetEngineReference_3.1.1.pdf